Extranet - Developer Guide

2017-02-10

Orckestra, Europe
Nygårdsvej 16
DK-2100 Copenhagen
Phone +45 3915 7600

www.orckestra.com

C1 CMS

Contents

1 Introduction

This guide is intended for Web and C# developers who want to customize or extend the Extranet add-on as well as learn how its security works.

The Extranet add-on securely protects your websites by only allowing users registered in your extranet to access or view web pages and media not intended for general public access.

For information about using the Extranet add-on, please refer to the *Extranet User Guide*.

In this guide you will learn how to create custom extranet providers by integrating them with external databases as well as learn about the Extranet Facade methods that will allow you to programmatically manage as well as customize and extend the out-of-the-box Extranet add-on. You will also have an overview of security used in this add-on.

For creating custom extranet providers as well as using the Extranet Facade methods, you should be proficient in C#/.NET programming. Additionally you should know the API of a 3rd-party database application you want to integrate with Extranet via your custom extranet providers.

1.1 Before You Start

You should make sure that you have the following prerequisites in place before you start the guide:

- You have installed and configured the latest version of C1 CMS
- You are using a test C1 CMS installation that you can safely experiment with.
- You have installed and configured the latest version of The Extranet add-on on this test C1 CMS installation instance.

Important: *You should never use the production environment for learning purposes while trying the examples and following the instructions in this guide.*

2 Writing Custom Extranet Providers

By default, the Extranet add-on uses a built-in extranet provider. When you first open the Extranet perspective, it has neither users nor groups. Normally, you can add them manually and then assign users to groups or vice versa.

However, you are not limited to only one – default - extranet provider. For example, if you have your own database (an ERP or CRM system) with users you want to be your extranet users (customers, suppliers etc), you can create your own extranet provider that will get the users from this database, assign them to specific groups and use them instead of, or along with, the default extranet provider.

2.1 Default Extranet Provider

The Extranet add-on comes with one default extranet provider:

- DefaultExtranetProvider

The following assembly contains the code for this provider:

- Composite.Community.Extranet.dll

This assembly is installed in the Bin folder on an C1 CMS-based website.

The provider is plugged in via the C1 CMS configuration file (\\<*website*>*Apps_Data**Composite**Composite.config*) in the < **Composite.Community.Extranet.Plugins.ExtranetProviderConfiguration**> section:

```
<Composite.Community.Extranet.Plugins.ExtranetProviderConfiguration>
      <ExtranetProviderPlugins>
            <add name="Default" title="Extranet"
type="Composite.Community.Extranet.DefaultProvider.DefaultExtranetProvider,
Composite.Community.Extranet" />
      </ExtranetProviderPlugins>
</Composite.Community.Extranet.Plugins.ExtranetProviderConfiguration>
```

Listing 1: Default provider plugged in

Each <**add**> element stands for one extranet provider and has three mandatory attributes:

- name
- title
- type

The **name** attribute specifies the *name* of the extranet provider (This is the name by which the provider is referred to in the code.)

The **title** attribute specifies the *title* of the extranet provider. (This is the name by which the user identifies the extranet in the GUI).

The **type** attribute specifies the name of the extranet *provider class* followed by the provider's *assembly* separated by a comma.

In the default configuration, these values are as follows:

- **Name**: Default
- **Title**: Extranet
- **Provider Class**: Composite.Community.Extranet.DefaultProvider.DefaultExtranetProvider
- **Assembly**: Composite.Community.Extranet

2.2 Custom Extranet Providers

Using the plug-in model of The Extranet add-on, you can create custom extranet providers, integrate them into The Extranet add-on and use them on your websites.

These providers can get users and, possibly, groups, from external databases, for example, an ERP/CRM system or an SQL database.

Creating a custom provider requires creating an assembly (similar to the default one), which implements the plug-in model related classes and interfaces and plug it in to the Extranet add-on.

The steps to create a custom extranet provider include:

1. Creating a class library project and adding required references to it.
2. Creating required classes by implementing the Extranet plug-in model.
3. Building the provider and deploying it on a website.

In the following few sections, you will learn more about each of these steps.

2.2.1 Sample Code

For illustration, we will create a sample custom provider called “Simple Extranet Provider”.

To simplify the sample code, instead of using 3rd-party application APIs to retrieve users and groups from external sources, we will generate them in the custom provider on-the-fly and use them in our methods.

2.3 Creating Class Library

Each custom extranet provider is represented by a class library assembly. So you should start by creating a class library project by using a Class Library project template in Visual Studio 2008 (Visual C#, Windows, Class Library).

For our sample we will create the project called **SimpleExtranetProvider.**

2.3.1 Adding References

To be able to use C1 CMS, The Extranet add-on and other functionality, you should add a number of references to the project.Normally, you should add references to some of the assemblies located in the Bin folder of your website with the Extranet add-on already installed.

The following references must be added to the project (in addition to the default ones):

- Composite
- Composite.Community.Extranet
- Microsoft.Practices.EnterpriseLibrary.Common
- Microsoft.Practices.EnterpriseLibrary.Configuration.Design
- Microsoft.Practices.ObjectBuilder
- System.Configuration

Once you have created a Class Library project and added all required references, go on to create required classes for your extranet provider.

Before we start taking the steps to actually create a custom extranet provider, we have created a very simple “external” (dummy) source that we will use in our sample code to store and retrieve information about our users, groups and users-to-groups relations. You should use your own code to manage your external sources for this purpose.

2.3.2 Creating Simple External Source for Sample Code

For our sample code, we have created a simple “external” source to fill our extranet with new users and groups and store them in. This code is presented here for illustration only.

When you create your extranet provider, you will use a different source for your extranet members and should skip this step altogether.

In the extranet provider class that we will create shortly, we have two private lists to hold the information about the users and groups respectively that will serve as our “simple external source”.

```
private List<ExtranetUser> _users;
private List<ExtranetGroup> _groups;
```

We also have two dictionaries:

```
private Dictionary<Guid, List<Guid>> _groupUsers;
private Dictionary<string, string> _userName;
```

The first one holds relations between users and groups for us, the second one maps a username to a user password. The latter is absolutely unsafe in the real-life code. However, we use this simplistic approach for illustration only.

We have created a private method which takes care of initialization of those private variables and called it from the extranet provider class’s constructor to have our simple “external” source up and running.

```
private void CreateSimpleExternalSource()
{
  // generate 20 users
  _users = (from u in Enumerable.Range(1, 20)
            select new ExtranetUser
            {
              Id = Guid.NewGuid(),
              CreationDate = new DateTime(2009, 1, 1).AddDays(u),
              IsApproved = true,
              UserName = "User" + (u.ToString().Length > 1 ? "" : "0") +
u.ToString(),
              Email = "user" + (u.ToString().Length > 1 ? "" : "0") +
u.ToString() + "@dummymail.net",
              Name = "User " + (u.ToString().Length > 1 ? "" : "0") +
u.ToString(),
              FolderName = (u % 2 == 0) ? "Domestic Companies" : "Foreign
Companies",
              ProviderName = "Simple"
            }).ToList();

  // set passwords
  _userName = new Dictionary<string, string>();
  foreach (ExtranetUser user in _users)
  {
    _userName.Add(user.UserName, user.Name);
  }

  // create groups
  _groups = new List<ExtranetGroup>();

  // create the root group
  ExtranetGroup group = new ExtranetGroup();
  group.Id = Guid.NewGuid();
  group.Name = "All Groups";
  group.ParentGroupId = null;
  group.CanBeAssignedAccessRight = false;
  group.ProviderName = "Simple";

  Guid rootGroupId = group.Id;
  _groups.Add(group);

  // create the Customers sub-group
  group = new ExtranetGroup();
  group.Id = Guid.NewGuid();
  group.Name = "Customers";
  group.ParentGroupId = rootGroupId;
  group.CanBeAssignedAccessRight = true;
  group.MemberUsersUpdatable = true;
  group.ProviderName = "Simple";

  _groups.Add(group);

  // create the Suppliers sub-group
  group = new ExtranetGroup();
  group.Id = Guid.NewGuid();
  group.Name = "Suppliers";
  group.ParentGroupId = rootGroupId;
  group.CanBeAssignedAccessRight = true;
  group.MemberUsersUpdatable = true;
  group.ProviderName = "Simple";

  _groups.Add(group);

  // assign users to groups
  _groupUsers = new Dictionary<Guid, List<Guid>>();

  List<ExtranetUser>.Enumerator userEnum = _users.GetEnumerator();
```

```
  foreach (ExtranetGroup pg in _groups.Where(f =>
f.CanBeAssignedAccessRight == true))
  {
    List<Guid> userIds = new List<Guid>();
    for (int i = 0; i < 10; i++)
    {
      userEnum.MoveNext();
      userIds.Add(userEnum.Current.Id);
    }

    _groupUsers.Add(pg.Id, userIds);
  }
}
```

Listing 2: Generating a dummy source to populate the custom extranet

In the example above:

1. We generate 20 user accounts by setting the properties of the **ExtranetUser** class instances and adding these instances to the **_users** list. We name the users as "*User01*" to "*User20*", set their "dummy" email addresses and place them in one of the two folders: "*Domestic Companies*" and "*Foreign Companies*".
2. Then we set their passwords by using our **_userName** dictionary.
3. We continue to create one root group ("All Groups") and two sub-groups under this group – "*Customers*" and "*Suppliers*" - by using **ExtranetGroup** and adding their instances to the **_groups** list.
4. Finally, we assign one half of the user accounts to the *Customers* group and the other to the *Suppliers* group by using **_groupUsers**.

This is the structure of user accounts and user groups that will appear in the Extranet perspective out-of-the-box when we eventually build and deploy our sample Simple Extranet Provider.

2.4 Creating Required Classes

To integrate your extranet provider into the Extranet add-on, you should create a class that will represent this provider.

Based on your requirements to a custom provider you can choose to implement one or more interfaces when creating this class. All the interfaces that you should or might implement are located in the following namespace:

- Composite.Community.Extranet.Plugins.ExtranetProvider

You should at least implement one interface: IExtranetProvider.This interface provides the basic functionality needed for a custom provider to work. This interface gives read-only access to the external extranet user accounts and groups as well as the users-to-groups assignments.

To be able to edit these user accounts, groups and the relations between them, you should implement a number of other interfaces, which we will shortly cover.

Let's start with creating the custom extranet provider class by implementing this interface and then proceed to implementation of other useful interfaces.

2.4.1 Step 1: Implementing IExtranetProvider Interface

First of all, you should create the class for your custom extranet provider.

This class should at minimum:

- Provide the title of the extranet provider
- Retrieve a user by ID
- Retrieve the user ID by name
- Retrieve all the users
- Retrieve a group
- Retrieve group IDs assigned to one user
- Retrieve users assigned to one group
- Get the hierarchy of a group
- Lock out a user
- Set a user active

To create the extranet provider class, you should implement at least one interface: **IExtranetProvider.**

```
public interface IExtranetProvider
{
	string ProviderTitle { get; }
	Guid GetUserId(string userName);
	IExtranetUser GetUser(Guid userId);
	IEnumerable<IExtranetUser> GetAllUsers();
	IEnumerable<Guid> GetGroupIdListForUser(Guid userId);
	IEnumerable<IExtranetGroupHierarchyNode> GetGroupHierarchy();
	IExtranetGroup GetGroup(Guid groupId);
	IEnumerable<IExtranetUser> GetUsersFromGroup(Guid groupId);
	void SetUserIsLockedOut(Guid userId, bool lockedOut);
	void UserActivity(Guid userId);
}
```

Listing 3: IExtranetProvider interface

In the following example, we have created the **SimpleExtranetProvider** for our sample solution by implementing the **IExtranetProvider** interface. This sample also makes use of our simple “external” source we have created earlier for illustration to populate our custom extranet.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Composite.Community.Extranet.Plugins.ExtranetProvider;
using Composite.Community.Extranet.Data;
using Composite.Community.Extranet.DefaultProvider;
using Microsoft.Practices.EnterpriseLibrary.Common.Configuration;
using
Microsoft.Practices.EnterpriseLibrary.Common.Configuration.ObjectBuilder;
using System.Configuration;
using Microsoft.Practices.ObjectBuilder;
using Composite.Community.Extranet;
using Composite.Renderings.Page;

namespace SimpleExtranetProvider
{
  [ConfigurationElementType(typeof(SimpleExtranetProviderConfiguration))]
  public class SimpleExtranetProvider : IExtranetProvider
  {
    private string _providerTitle = null;

    private List<ExtranetUser> _users;
    private List<ExtranetGroup> _groups;
    private Dictionary<Guid, List<Guid>> _groupUsers;
    private Dictionary<string, string> _userName;

    public SimpleExtranetProvider()
      : this("Simple Extranet")
    {
    }

    public SimpleExtranetProvider(string providerTitle)
    {
      _providerTitle = providerTitle;

       CreateSimpleExternalSource();
    }

    #region IExtranetProvider Members

    public string ProviderTitle
    {
      get
      {
        return _providerTitle;
      }
    }

    public Guid GetUserId(string userName)
    {
      IExtranetUser user = _users.Where(f => f.UserName ==
userName).FirstOrDefault();
      if (user == null)
        return Guid.Empty;
      else
        return user.Id;
    }

    public IExtranetUser GetUser(Guid userId)
    {
      return _users.Where(f => f.Id == userId).FirstOrDefault();
    }

    public IEnumerable<IExtranetUser> GetAllUsers()
    {
      return _users.Select(f => f as IExtranetUser);
```

```
    }

    public IEnumerable<Guid> GetGroupIdListForUser(Guid userId)
    {
      foreach (Guid groupId in _groupUsers.Keys)
      {
        if (_groupUsers[groupId].Contains(userId))
          yield return groupId;
      }
    }

    public IEnumerable<IExtranetGroupHierarchyNode> GetGroupHierarchy()
    {
      foreach (ExtranetGroup gp in _groups.Where(f => f.ParentGroupId ==
null))
      {
        ExtranetGroupHierarchyNode ghn = new
ExtranetGroupHierarchyNode(gp);
        ghn.Groups = GetSubGroupHierarchy(gp.Id);
        yield return ghn;
      }
    }

    private IEnumerable<IExtranetGroupHierarchyNode>
GetSubGroupHierarchy(Guid groupId)
    {
      foreach (ExtranetGroup gp in _groups.Where(f => f.ParentGroupId ==
groupId))
      {
        ExtranetGroupHierarchyNode ghn = new
ExtranetGroupHierarchyNode(gp);
        ghn.Groups = GetSubGroupHierarchy(gp.Id);
        yield return ghn;
      }
    }

    public IExtranetGroup GetGroup(Guid groupId)
    {
      return _groups.Where(f => f.Id == groupId).First();
    }

    public IEnumerable<IExtranetUser> GetUsersFromGroup(Guid groupId)
    {
      if (_groupUsers.ContainsKey(groupId))
      {
        return _groupUsers[groupId].Select(f => _users.Where(n => n.Id ==
f).First() as IExtranetUser);
      }
      else
      {
        return new List<IExtranetUser>();
      }
    }

    public void SetUserIsLockedOut(Guid userId, bool lockedOut)
    {
      this._users[this._users.FindIndex(f => f.Id == userId)].IsLockedOut =
lockedOut;
      this._users[this._users.FindIndex(f => f.Id ==
userId)].LastLockedOutDate = DateTime.Now;
    }

    public void UserActivity(Guid userId)
    {
      this._users[this._users.FindIndex(f => f.Id ==
userId)].LastActivityDate = DateTime.Now;
    }
```

```
    #endregion

    #region External Source
    private void CreateSimpleExternalSource()
    {
      // generate 20 users
      _users = (from u in Enumerable.Range(1, 20)
            select new ExtranetUser
            {
              Id = Guid.NewGuid(),
              CreationDate = new DateTime(2009, 1, 1).AddDays(u),
              IsApproved = true,
              UserName = "User" + (u.ToString().Length > 1 ? "" : "0") +
u.ToString(),
              Email = "user" + (u.ToString().Length > 1 ? "" : "0") +
u.ToString() + "@dummymail.net",
              Name = "User " + (u.ToString().Length > 1 ? "" : "0") +
u.ToString(),
              FolderName = (u % 2 == 0) ? "Domestic Companies" : "Foreign
Companies",
              ProviderName = "Simple"
            }).ToList();

      // set passwords
      _userName = new Dictionary<string, string>();
      foreach (ExtranetUser user in _users)
      {
        _userName.Add(user.UserName, user.Name);
      }

      // create groups
      _groups = new List<ExtranetGroup>();

      // create the root group
      ExtranetGroup group = new ExtranetGroup();
      group.Id = Guid.NewGuid();
      group.Name = "All Groups";
      group.ParentGroupId = null;
      group.CanBeAssignedAccessRight = false;
      group.ProviderName = "Simple";

      Guid rootGroupId = group.Id;
      _groups.Add(group);

      // create the Customers sub-group
      group = new ExtranetGroup();
      group.Id = Guid.NewGuid();
      group.Name = "Customers";
      group.ParentGroupId = rootGroupId;
      group.CanBeAssignedAccessRight = true;
      group.MemberUsersUpdatable = true;
      group.ProviderName = "Simple";

      _groups.Add(group);

      // create the Suppliers sub-group
      group = new ExtranetGroup();
      group.Id = Guid.NewGuid();
      group.Name = "Suppliers";
      group.ParentGroupId = rootGroupId;
      group.CanBeAssignedAccessRight = true;
      group.MemberUsersUpdatable = true;
      group.ProviderName = "Simple";

      _groups.Add(group);
```

```
      // assign users to groups
      _groupUsers = new Dictionary<Guid, List<Guid>>();

      List<ExtranetUser>.Enumerator userEnum = _users.GetEnumerator();
      foreach (ExtranetGroup pg in _groups.Where(f =>
f.CanBeAssignedAccessRight == true))
      {
        List<Guid> userIds = new List<Guid>();
        for (int i = 0; i < 10; i++)
        {
          userEnum.MoveNext();
          userIds.Add(userEnum.Current.Id);
        }

        _groupUsers.Add(pg.Id, userIds);
      }
    }
    #endregion
  }


  [Assembler(typeof(SimpleExtranetProviderAssembler))]
  public class SimpleExtranetProviderConfiguration : ExtranetProviderData
  {
    private const string _titleProperty = "title";
    [ConfigurationProperty(_titleProperty, IsRequired = true)]
    public string Title
    {
      get
      {
        return (string)base[_titleProperty];
      }
    }
  }

  internal sealed class SimpleExtranetProviderAssembler :
IAssembler<IExtranetProvider, ExtranetProviderData>
  {
    public IExtranetProvider Assemble(IBuilderContext context,
ExtranetProviderData objectConfiguration, IConfigurationSource
configurationSource, ConfigurationReflectionCache reflectionCache)
    {
      SimpleExtranetProviderConfiguration config = objectConfiguration as
SimpleExtranetProviderConfiguration;

      SimpleExtranetProvider provider = new
SimpleExtranetProvider(config.Title);

      return provider;
    }
  }
}
```

Listing 4: Implementing IExtranetProvider in the sample class

As you can see in the example above:

1. For our simple extranet provider, in the constructor we call **CreateSimpleExtrenalSource** method that populates our custom extranet with user accounts, groups etc.
2. Each implemented method returns the required value by retrieving it from our “external” source.
3. In the constructor, we also initialize the provider’s title retrieved from the configuration file.

4. Since the provider's *title* is retrieved from the **Composite.config** file, we implement the required logic to get the provider's title (the **ConfigurationElementType** attribute and the **SimpleExtranetProviderConfiguration** class).

This sample code is self-sufficient. If you build, deploy and plug it in, you will have a working instance of a custom extranet provider in read-only mode. You will be able to only view the extranet from the Extranet perspective and use it on your websites and media.

We will go a little further and implement a few other interfaces to allow administrators to add, delete and modify users and groups, assign the users to groups and groups to users, change passwords and validate users on extranet protected websites and media by granting or denying access to web pages and media files.

2.4.2 Step 2: Implementing IExtranetUserUpdateProvider Interface

To allow extranet administrators to add, modify and delete extranet user accounts from the CMS Console, you should implement the **IExtranetUserUpdateProvider** interface.

By implementing this interface in your custom extranet provider class, you also enable the administrators to assign users to groups and groups to users as well as (re)set the users' passwords.

```
public interface IExtranetUserUpdateProvider
{
	IExtranetUser AddNewUser(IExtranetUser user);
	void UpdateUser(IExtranetUser user);
	void SetGroupsForUser(Guid userId, IEnumerable<Guid> groupIdList);
	void SetUsersForGroup(Guid groupId, IEnumerable<Guid> userIdList);
	bool SetUserPassword(Guid userId, string newPassword);
	void DeleteUser(Guid userId);
}
```

Listing 5: IExtranetUserUpdateProvider interface

2.4.3 Step 3: Implementing IExtranetGroupUpdateProvider Interface

To allow extranet administrators to add, modify and delete extranet user groups, you should implement the **IExtranetGroupUpdateProvider** interface.

```
public interface IExtranetGroupUpdateProvider
{
	IExtranetGroup AddNewGroup(IExtranetGroup group);
	void UpdateGroup(IExtranetGroup group);
	void DeleteGroup(Guid groupId);
}
```

Listing 6: IExtranetGroupUpdateProvider interface

2.4.4 Step 4: Implementing IExtranetUserPasswordProvider Interface

Allowing users to restore their forgotten passwords via email requires implementation of the **IExtranetUserPasswordProvider** interface.

```
public interface IExtranetUserPasswordProvider
{
	string GetUserPassword(string login);
}
```

Listing 7: IExtranetUserPasswordProvider interface

2.4.5 Step 5: Implementing IExtranetFormsLoginProvider Interface

To validate users on protected website resources, you should implement the **IExtranetFormsLoginProvider** interface.

```
public interface IExtranetFormsLoginProvider
{
	bool ValidateUser(string login, string password, out Guid userId);
}
```

Listing 8: IExtranetFormsLoginProvider interface

2.4.6 Finalizing Simple Extranet Provider Class

In the following example, we have additionally implemented the **IExtranetUserUpdateProvider**, **IExtranetGroupUpdateProvider**, **IExtranetUserPasswordProvider** and **IExtranetFormsLoginProvider** interfaces in our **SimpleExtranetProvider** class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Composite.Community.Extranet.Plugins.ExtranetProvider;
using Composite.Community.Extranet.Data;
using Composite.Community.Extranet.DefaultProvider;
using Microsoft.Practices.EnterpriseLibrary.Common.Configuration;
using
Microsoft.Practices.EnterpriseLibrary.Common.Configuration.ObjectBuilder;
using System.Configuration;
using Microsoft.Practices.ObjectBuilder;
using Composite.Community.Extranet;
using Composite.Renderings.Page;

namespace SimpleExtranetProvider
{
  [ConfigurationElementType(typeof(SimpleExtranetProviderConfiguration))]
  public class SimpleExtranetProvider : IExtranetProvider,
IExtranetFormsLoginProvider, IExtranetUserPasswordProvider,
IExtranetUserUpdateProvider, IExtranetGroupUpdateProvider
  {
    private string _providerTitle = null;

    private List<ExtranetUser> _users;
    private List<ExtranetGroup> _groups;
    private Dictionary<Guid, List<Guid>> _groupUsers;
    private Dictionary<string, string> _userName;

    public SimpleExtranetProvider()
      : this("Simple Extranet")
    {
    }

    public SimpleExtranetProvider(string providerTitle)
    {
      _providerTitle = providerTitle;

       CreateSimpleExternalSource();
    }

    #region IExtranetProvider Members

    public string ProviderTitle
    {
      get
      {
        return _providerTitle;
      }
    }

    public Guid GetUserId(string userName)
    {
      IExtranetUser user = _users.Where(f => f.UserName ==
userName).FirstOrDefault();
      if (user == null)
        return Guid.Empty;
      else
        return user.Id;
    }

    public IExtranetUser GetUser(Guid userId)
    {
      return _users.Where(f => f.Id == userId).FirstOrDefault();
    }

    public IEnumerable<IExtranetUser> GetAllUsers()
```

```
    {
      return _users.Select(f => f as IExtranetUser);
    }

    public IEnumerable<Guid> GetGroupIdListForUser(Guid userId)
    {
      foreach (Guid groupId in _groupUsers.Keys)
      {
        if (_groupUsers[groupId].Contains(userId))
          yield return groupId;
      }
    }

    public IEnumerable<IExtranetGroupHierarchyNode> GetGroupHierarchy()
    {
      foreach (ExtranetGroup gp in _groups.Where(f => f.ParentGroupId ==
null))
      {
        ExtranetGroupHierarchyNode ghn = new
ExtranetGroupHierarchyNode(gp);
        ghn.Groups = GetSubGroupHierarchy(gp.Id);
        yield return ghn;
      }
    }

    private IEnumerable<IExtranetGroupHierarchyNode>
GetSubGroupHierarchy(Guid groupId)
    {
      foreach (ExtranetGroup gp in _groups.Where(f => f.ParentGroupId ==
groupId))
      {
        ExtranetGroupHierarchyNode ghn = new
ExtranetGroupHierarchyNode(gp);
        ghn.Groups = GetSubGroupHierarchy(gp.Id);
        yield return ghn;
      }
    }

    public IExtranetGroup GetGroup(Guid groupId)
    {
      return _groups.Where(f => f.Id == groupId).First();
    }

    public IEnumerable<IExtranetUser> GetUsersFromGroup(Guid groupId)
    {
      if (_groupUsers.ContainsKey(groupId))
      {
        return _groupUsers[groupId].Select(f => _users.Where(n => n.Id ==
f).First() as IExtranetUser);
      }
      else
      {
        return new List<IExtranetUser>();
      }
    }

    public void SetUserIsLockedOut(Guid userId, bool lockedOut)
    {
      this._users[this._users.FindIndex(f => f.Id == userId)].IsLockedOut =
lockedOut;
      this._users[this._users.FindIndex(f => f.Id ==
userId)].LastLockedOutDate = DateTime.Now;
    }

    public void UserActivity(Guid userId)
    {
```

```
      this._users[this._users.FindIndex(f => f.Id ==
userId)].LastActivityDate = DateTime.Now;
    }

    #endregion

    #region IExtranetUserPasswordProvider Members

    public string GetUserPassword(string login)
    {
      return _userName[login];
    }

    #endregion

    #region IExtranetFormsLoginProvider Members

    public bool ValidateUser(string login, string password, out Guid
userId)
    {
      userId = Guid.Empty;
      if (_userName.ContainsKey(login))
      {
        if (_userName[login] == password)
        {
          userId = _users.Where(f => f.UserName ==
login).FirstOrDefault().Id;

          Guid pageId = PageRenderer.CurrentPageId;
          IEnumerable<Guid> userGroupIds = GetGroupIdListForUser(userId);
          IEnumerable<Guid> pageGroupIds =
ExtranetFacade.GetPageSecurityGroups("Simple", pageId);
          if (ExtranetFacade.ValidateUserGroupsToItemGroups("Simple",
userGroupIds, pageGroupIds))
          {
            Guid lampdaUserId = userId;
            _users[_users.FindIndex(f => f.Id ==
lampdaUserId)].LastLoginDate = DateTime.Now;
            _users[_users.FindIndex(f => f.Id ==
lampdaUserId)].LastActivityDate = DateTime.Now;
            return true;
          }
        }
      }
      return false;
    }

    #endregion

    #region IExtranetUserUpdateProvider Members

    public IExtranetUser AddNewUser(IExtranetUser user)
    {
      _users.Add(user as ExtranetUser);
      return user;
    }

    public void UpdateUser(IExtranetUser user)
    {
      _users[_users.FindIndex(f => f.Id == user.Id)] = (ExtranetUser)user;
    }

    public void DeleteUser(Guid userId)
    {
      ExtranetUser user = _users.Where(f => f.Id == userId).First();
      _userName.Remove(user.UserName);
      _users.Remove(user);
```

```
      foreach (List<Guid> userlist in _groupUsers.Values)
      {
        userlist.Remove(userId);
      }
    }

    public bool SetUserPassword(Guid userId, string newPassword)
    {
      try
      {
        _userName[_users.Where(f => f.Id == userId).First().UserName] =
newPassword;
        _users[_users.FindIndex(f => f.Id ==
userId)].LastPasswordChangeDate = DateTime.Now;
        return true;
      }
      catch (Exception)
      {
        return false;
      }
    }

    public void SetGroupsForUser(Guid userId, IEnumerable<Guid>
groupIdList)
    {
      foreach (Guid groupId in _groupUsers.Keys)
      {
        if (_groupUsers[groupId].Contains(userId))
          _groupUsers[groupId].Remove(userId);
      }
      foreach (Guid groupId in groupIdList)
      {
        if (_groupUsers.ContainsKey(groupId))
          _groupUsers[groupId].Add(userId);
        else
          _groupUsers.Add(groupId, new List<Guid>() { userId });
      }
    }

    public void SetUsersForGroup(Guid groupId, IEnumerable<Guid>
userIdList)
    {
      if (_groupUsers.ContainsKey(groupId))
        _groupUsers[groupId] = userIdList.ToList();
      else
        _groupUsers.Add(groupId, userIdList.ToList());
    }
    #endregion

    #region IExtranetGroupUpdateProvider Members

    public IExtranetGroup AddNewGroup(IExtranetGroup group)
    {
      _groups.Add(new ExtranetGroup(group));
      return group;
    }

    public void DeleteGroup(Guid groupId)
    {
      _groups.RemoveAll(f => f.Id == groupId);
      _groupUsers.Remove(groupId);
    }

    public void UpdateGroup(IExtranetGroup group)
    {
      _groups[_groups.FindIndex(f => f.Id == group.Id)] = new
ExtranetGroup(group);
```

```
    }

    #endregion

    #region External Source
    private void CreateSimpleExternalSource()
    {
      // generate 20 users
      _users = (from u in Enumerable.Range(1, 20)
            select new ExtranetUser
            {
              Id = Guid.NewGuid(),
              CreationDate = new DateTime(2009, 1, 1).AddDays(u),
              IsApproved = true,
              UserName = "User" + (u.ToString().Length > 1 ? "" : "0") +
u.ToString(),
              Email = "user" + (u.ToString().Length > 1 ? "" : "0") +
u.ToString() + "@dummymail.net",
              Name = "User " + (u.ToString().Length > 1 ? "" : "0") +
u.ToString(),
              FolderName = (u % 2 == 0) ? "Domestic Companies" : "Foreign
Companies",
              ProviderName = "Simple"
            }).ToList();

      // set passwords
      _userName = new Dictionary<string, string>();
      foreach (ExtranetUser user in _users)
      {
        _userName.Add(user.UserName, user.Name);
      }

      // create groups
      _groups = new List<ExtranetGroup>();

      // create the root group
      ExtranetGroup group = new ExtranetGroup();
      group.Id = Guid.NewGuid();
      group.Name = "All Groups";
      group.ParentGroupId = null;
      group.CanBeAssignedAccessRight = false;
      group.ProviderName = "Simple";

      Guid rootGroupId = group.Id;
      _groups.Add(group);

      // create the Customers sub-group
      group = new ExtranetGroup();
      group.Id = Guid.NewGuid();
      group.Name = "Customers";
      group.ParentGroupId = rootGroupId;
      group.CanBeAssignedAccessRight = true;
      group.MemberUsersUpdatable = true;
      group.ProviderName = "Simple";

      _groups.Add(group);

      // create the Suppliers sub-group
      group = new ExtranetGroup();
      group.Id = Guid.NewGuid();
      group.Name = "Suppliers";
      group.ParentGroupId = rootGroupId;
      group.CanBeAssignedAccessRight = true;
      group.MemberUsersUpdatable = true;
      group.ProviderName = "Simple";

      _groups.Add(group);
```

```
      // assign users to groups
      _groupUsers = new Dictionary<Guid, List<Guid>>();

      List<ExtranetUser>.Enumerator userEnum = _users.GetEnumerator();
      foreach (ExtranetGroup pg in _groups.Where(f =>
f.CanBeAssignedAccessRight == true))
      {
        List<Guid> userIds = new List<Guid>();
        for (int i = 0; i < 10; i++)
        {
          userEnum.MoveNext();
          userIds.Add(userEnum.Current.Id);
        }

        _groupUsers.Add(pg.Id, userIds);
      }
    }
    #endregion
  }

  [Assembler(typeof(SimpleExtranetProviderAssembler))]
  public class SimpleExtranetProviderConfiguration : ExtranetProviderData
  {
    private const string _titleProperty = "title";
    [ConfigurationProperty(_titleProperty, IsRequired = true)]
    public string Title
    {
      get
      {
        return (string)base[_titleProperty];
      }
    }
  }

  internal sealed class SimpleExtranetProviderAssembler :
IAssembler<IExtranetProvider, ExtranetProviderData>
  {
    public IExtranetProvider Assemble(IBuilderContext context,
ExtranetProviderData objectConfiguration, IConfigurationSource
configurationSource, ConfigurationReflectionCache reflectionCache)
    {
      SimpleExtranetProviderConfiguration config = objectConfiguration as
SimpleExtranetProviderConfiguration;

      SimpleExtranetProvider provider = new
SimpleExtranetProvider(config.Title);

      return provider;
    }
  }

}
```

Listing 9: Implementing four other interfaces in the sample class

In the example above:

1. We implement methods of all the four interfaces and thus extend our custom provider's capabilities. In each case, we simply use our built-in simple "external" source to manage user accounts, groups, passwords and users-to-groups relations.
2. In the **ValidateUser** method called when a user tries to access protected resources, we not only check if the user account and the password are valid but also check the user account's access permissions to the current page.

3. We do the latter by getting the list of groups the user is member of and the list of groups allowed on the resource (a page, in our case) and then calling the **ExtranetFacade**'s method **ValidateUserGroupsToItemGroups** passing the lists as its parameters
4. Unlike our sample code, the real code should additionally validate the user's groups against the so-called inherited groups by using the ExtranetFacade's **GetPageSecurityInheritedGroups** method along with **GetPageSecurityGroups** used in our sample code.
5. Unlike in the sample code, in the real-life code you will not only validate the user against access permissions to pages but also other items such as media.

Note: *You will learn more about the* ***ExtranetFacade*** *methods in the following chapter.*

Now that you have created your custom extranet provider, go on to build and deploy it on your website.

2.5 Building and Deploying

Once you have built your solution, you are ready to deploy and use it on your website.

To deploy the solution, you can follow one of the two approaches:

- Automatic
- Manual

For **automatic** deployment, you should build an add-on for your extranet provider and then install it on your C1 CMS via its Packages system.

For **manual** deployment, you should copy the assembly you have just built to a specific folder on your website and plug it in via the C1 CMS configuration file.

2.5.1 Automatic Deployment

For automatic deployment, you should build an add-on for your extranet provider following the standard add-on-building procedure.

In the *Install.xsl* file, you should specify the name of your custom provider, its class and its assembly.

The following is the sample for the **SimpleExtranetProvider** we have created:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
	<xsl:template match="@* | node()">
		<xsl:copy>
			<xsl:apply-templates select="@* | node()" />
		</xsl:copy>
	</xsl:template>
	<xsl:template match="/configuration/
Composite.Community.Extranet.Plugins.ExtranetProviderConfiguration/
ExtranetProviderPlugins">
		<xsl:copy>
			<xsl:apply-templates select="@* | node()" />
			<xsl:if test="count(add[@name='Simple'])=0">
				<add name="Simple" title="Simple Extranet"
type="SimpleExtranetProvider.SimpleExtranetProvider,
SimpleExtranetProvider" />
			</xsl:if>
		</xsl:copy>
	</xsl:template>
</xsl:stylesheet>
```

Listing 10: Sample Install.xsl for SimpleExtranetProvider

In the similar way, you should modify the *Uninstall.xsl*.

Once you have built the add-on, you can install it via the Packages system in the CMS Console (**System** > **Packages** > **Installed Packages** > **Local Packages** > **Install local package**).

2.5.2 Manual Deployment

To deploy the custom extranet provider manually:

1. Copy the extranet provider assembly to the *Bin* subfolder in the root folder of your website.
2. Open the C1 CMS configuration file found at *\\<website>\App_Data\Composite\Composite.config*

3. Locate the <**Composite.Community.Extranet.Plugins.ExtranetProviderConfiguration**> section.

4. Under the <**ExtranetProviderPlugins**> element add the name and title of your provider, its class and its assembly.

   The following is the sample for the **SimpleExtranetProvider** we have created:

```
<add name="Simple" title="Simple Extranet"
type="SimpleExtranetProvider.SimpleExtranetProvider,
SimpleExtranetProvider" />
```

Listing 11: Sample of plugging in `SimpleExtranetProvider`

5. Now restart the server and then refresh the browser window (or the tab) in which you have your CMS Console running.

Now that you have deployed the custom extranet provider, you can start using it.

2.6 Using Custom Extranet in C1 CMS

Once you have deployed your custom extranet provider, it will appear in the Extranet perspective as another extranet.

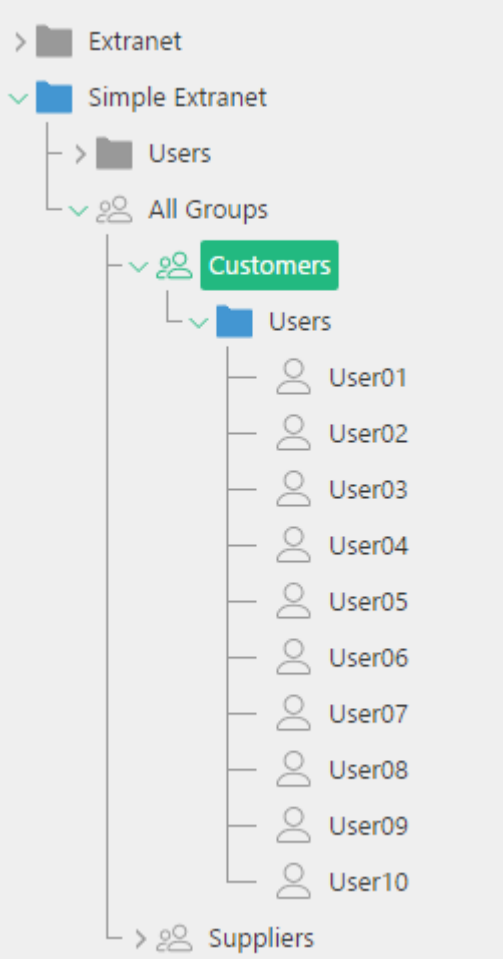

Figure 1: Custom extranet

This extranet presents the users retrieved from your database and assigned to specific groups in your custom extranet.

You can use this custom extranet to protect your website and media folders and manage extranet security on web pages and media files as you do with the default extranet provider.

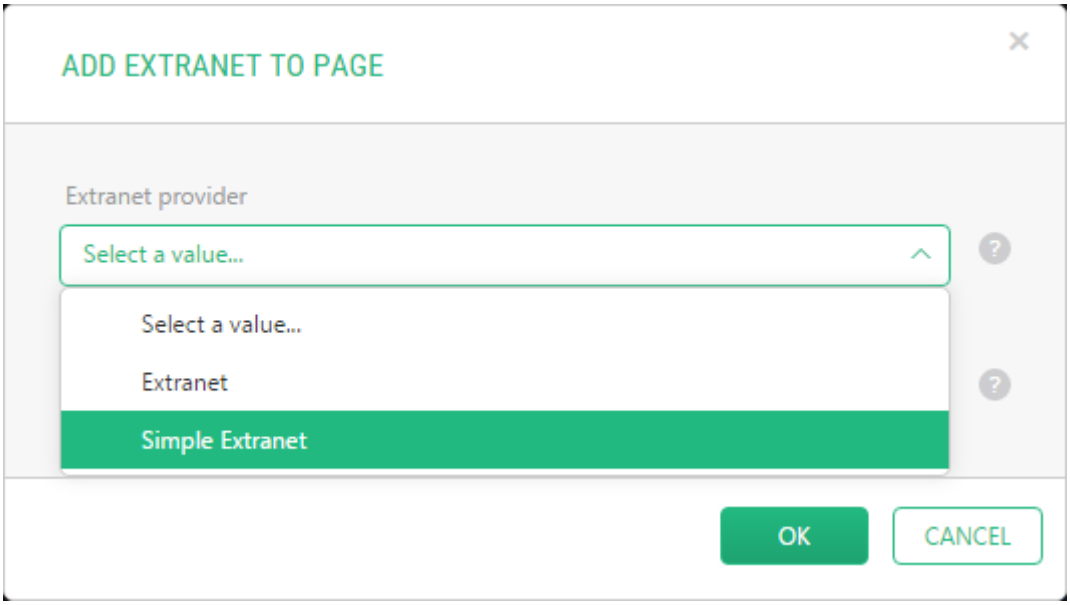

Figure 2: Adding the custom extranet to a website

3 Using Extranet Facade Methods

ExtranetFacade allows you to access and use almost all the methods that manage the Extranet functionality. By using the **ExtranetFacade** methods you can programmatically manage user accounts, user groups, assign users to groups and vice versa, add extranet security to websites and media folders as well as assign groups to pages and media files. You can also retrieve information about the extranet provider and manage logins and user validation on the extranet.

The public static **ExtranetFacade** class is part of the **Composite.Community.Extranet.ExtranetFacade** namespace.

You can use the **ExtranetFacade** methods when programmatically managing the default extranet provider as well as when developing your custom extranet providers.

The **ExtranetFacade** methods can be grouped as follows:

- Managing extranet user accounts
- Managing extranet user groups
- Managing users-to-groups relations
- Managing extranet on websites
- Managing extranet on media folders
- Managing extranet provider settings
- Managing logins and validation

In the following few sections, you will learn more about methods in each of these groups.

3.1 Managing Extranet User Accounts

Managing the users includes creating and deleting user accounts in the extranet as well as getting and setting their properties, for example, passwords or activity statuses.

In most cases, you should have the name of the Extranet provider available, before you call any of these methods.

Often, before adding or updating a user account, you should create or get an instance of the **IExtranetUser** and set a number of its properties.

To get or update specific properties of a specific user account, you should use its user ID.

3.1.1 IExtranetUser

IExtranetUser holds the properties that provide the basic information about the extranet user account as well as allow you to set or update some of its values when you create or update a user account. Therefore, some of the values are assigned internally and are read-only while others can be changed in the code.

For example, the user ID is automatically assigned by the system whenever a new user account is created. All the date-based properties are also handled by the system and you can only read their values.

```
public interface IExtranetUser
{
    string Comment { get; set; }
    DateTime CreationDate { get; }
    string Email { get; set; }
    string FolderName { get; set; }
    Guid Id { get; }
    bool IsApproved { get; set; }
    bool IsLockedOut { get; }
    DateTime LastActivityDate { get; }
    DateTime LastLockedOutDate { get; }
    DateTime LastLoginDate { get; }
    DateTime LastPasswordChangeDate { get; }
    string Name { get; set; }
    string ProviderName { get; }
    string UserName { get; set; }
}
```

Properties

Id	The user ID (read-only)
Name	The user's name (normally, the first and the last name)
UserName	The username used by the user to log in to the extranet
Email	The user's email address
FolderName	The folder where the user is placed
Comment	Comment to the user account
IsApproved	An indication of whether the user is active
IsLockedOut	An indication of whether the user is locked out (read-only)
CreationDate	The date the user account was created (read-only)
LastActivityDate	The last date the user's status was set to active (read-only)

LastLockedOutDate	The last date the user account was locked out (read-only)
LastLoginDate	The last date the user logged in to the extranet (read-only)
LastPasswordChangeDate	The last date the user's password was changed (read-only)
ProviderName	The extranet provider's name

3.1.2 AddNewUser

This method adds a new user to the extranet identified by its provider's name. Before adding the user, you should initialize an **IExtranetUser** object setting its required properties.

```
public static IExtranetUser AddNewUser(string providerName, IExtranetUser
user);
```

Parameters

providerName	The name of the Extranet provider
user	An **IExtranetUser** object

Return Value

The **IExtranetUser** object if the operation succeeds; otherwise, null.

3.1.3 GetUser (1)

This method retrieves the extranet user account as the **IExtranetUser** object if it can be found by its user ID in the extranet identified by its provider's name. You can further query or use its specific properties.

```
public static IExtranetUser GetUser(string providerName, Guid userId);
```

Parameters

providerName	The name of the Extranet provider
userId	The user ID.

Return Value

The **IExtranetUser** object if the operation succeeds; otherwise, null.

3.1.4 GetUser (2)

This method retrieves the extranet user account as the **IExtranetUser** object if it can be found by its user ID in one of the extranets used on any existing website. You can further query or use its specific properties.

```
public static IExtranetUser GetUser(Guid userId);
```

Parameters

userId	The user ID.

Return Value

The **IExtranetUser** object if the operation succeeds; otherwise, null.

3.1.5 GetAllUsers

This method retrieves all extranet user accounts as a collection of **IExtranetUser** objects in the extranet identified by its provider's name.

```
public static IEnumerable<IExtranetUser> GetAllUsers(string providerName);
```

Parameters

providerName	The name of the Extranet provider

Return Value

A collection of **IExtranetUser** objects that can be iterated.

3.1.6 GetUserId

This method gets the user's ID by the login name (username) in the extranet identified by its provider's name.

```
public static Guid GetUserId(string providerName, string login);
```

Parameters

providerName	The name of the Extranet provider
login	The name used to log in to the extranet (login, username)

Return Value

The GUID assigned to this user account.

3.1.7 GetUserPassword

This method gets the user's password by the login name (username) in the extranet identified by its provider's name.

```
public static string GetUserPassword(string providerName, string login);
```

Parameters

providerName	The name of the Extranet provider
login	The username used to log in to the extranet

Return Value

A string that holds the user's password.

3.1.8 GetCurrentUser

This method returns the current user of the extranet identified by its provider's name.

```
public static IExtranetUser GetCurrentUser(string providerName);
```

Parameters

providerName	The name of the extranet provider the current user account of which is retrieved.

Return Value

An **IExtranetUser** object if there is the current user in the extranet; otherwise, null.

3.1.9 GetCurrentUserId

This method returns the current user's ID.

```
public static Guid GetCurrentUserId();
```

Return Value

A GUID that specifies the current user's ID.

3.1.10 GetCurrentUserName

This method returns the name of the current user of the extranet identified by its provider's name.

```
public static string GetCurrentUserName(string providerName);
```

Parameters

providerName	The name of the extranet provider the name of the current user account of which is retrieved.

Return Value

A string that specifies the current extranet user's name.

3.1.11 UpdateUser

This method updates settings of an existing user account in the extranet identified by its provider's name. Before updating the user account's settings, you should get the existing user account as an **IExtranetUser** object and modify the desired properties with new values.

```
public static void UpdateUser(string providerName, IExtranetUser user);
```

Parameters

providerName	The name of the Extranet provider.
user	An **IExtranetUser** object

3.1.12 SetUserPassword

This method (re)sets the password of an existing user account by the user ID in the extranet identified by its provider's name.

```
public static bool SetUserPassword(string providerName, Guid userId, string
newPassword);
```

Parameters

providerName	The name of the Extranet provider.
userId	The user's ID.
newPassword	The new password for the user

Return Value

If the password is successfully changed, it returns true; otherwise, false.

3.1.13 UserActivity

This method sets the user account's status to active in the extranet identified by its provider's name. Unless the user account is locked out, the user can log on to the extranet with his or her credentials. The user account is identified by ist user ID.

```
public static void UserActivity(string providerName, Guid userId);
```

Parameters

providerName	The name of the Extranet provider.
userId	The user's ID.

3.1.14 SetUserIsLockedOut

This method locks out the existing user account in the extranet identified by its provider's name. If the user account is locked out, the user cannot log on to the extranet. The user account is identified by its user ID.

```
public static void SetUserIsLockedOut(string providerName, Guid userId,
bool lockedOut);
```

Parameters

providerName	The name of the Extranet provider.
userId	The user's ID.
lockedOut	If *true*, the user account gets locked out; otherwise, unlocked out.

3.1.15 Deleting Users

This method deletes an existing user account by the user ID in the extranet identified by its provider's name.

```
public static void DeleteUser(string providerName, Guid userId);
```

Parameters

providerName	The name of the Extranet provider.
userId	The user's ID.

3.2 Managing Extranet User Groups

Like managing extranet user accounts, managing the user groups includes creating and deleting groups in the extranet as well as getting and setting their properties, for example, their names or their permissions.

In most cases, you should have the name of the Extranet provider available, before you call any of these methods.

Often, before adding or updating a user account, you should create or get an instance of the **IExtranetGroup** and set a number of its properties.

To get or update specific properties of a specific user group, you should use its group ID.

3.2.1 Extranet Group

IExtranetGroup provides access to the properties of an extranet user group as well as allow you to set or update some of its values when you create or update a user group. The group ID is automatically assigned by the system whenever a new user group is created and is thus read-only.

```
public interface IExtranetGroup
{
    bool CanBeAssignedAccessRight { get; set; }
    string Description { get; set; }
    Guid Id { get; }
    bool MemberGroupsUpdatable { get; set; }
    bool MemberUsersUpdatable { get; set; }
    string Name { get; set; }
    Guid? ParentGroupId { get; set; }
    string ProviderName { get; set; }
}
```

Properties

Id	The group ID (read-only)
ParentGroupId	The parent group's ID. If null, the group is a root group.
Name	The group's name
Description	The group's description
MemberUsersUpdatable	If true, users can be added to, or deleted from, this group
MemberGroupsUpdatable	If true, sub-groups can be added to, or deleted from, this group
CanBeAssignedAccessRights	If true, this group can be used to grant or revoke access on websites and pages and media folders and files
ProviderName	The extranet provider's name

3.2.2 Extranet Group Hierarchy Node

IExtranetGroupHierarchyNode extends **IExtranetGroup** by providing an enumerator to the collection of sub-groups (if any) in the group as their parent group.

By using **IExtranetGroupHierarchyNode** you can access not only properties of the current user groups but also those of all its sub-groups if any.

```
public interface IExtranetGroupHierarchyNode : IExtranetGroup
{
    IEnumerable<IExtranetGroupHierarchyNode> Groups { get; }
}
```

Properties

Groups	An enumerable collection of sub- groups as IExtranetGroupHierarchyNode objects

3.2.3 AddNewGroup

This method adds a new group to the extranet identified by its provider's name. Before adding the group, you should initialize an **IExtranetGroup** object setting the required properties.

```
public static IExtranetGroup AddNewGroup(string providerName,
IExtranetGroup group);
```

Parameters

providerName	The name of the Extranet provider
group	An initialized IExtranetGroup object

Return Value

The **IExtranetGroup** object if the operation succeeds; otherwise, null.

3.2.4 GetGroup

This method retrieves the extranet user group as the **IExtranetGroup** object by its group ID, which allows you to further query or use its specific properties. It requires the extranet provider's name.

```
public static IExtranetGroup GetGroup(string providerName, Guid groupId);
```

Parameters

providerName	The name of the Extranet provider.
groupId	The group's ID.

Return Value

The **IExtranetGroup** object if the operation succeeds; otherwise, null.

3.2.5 GetRootGroup

This method retrieves the root group of an existing group by the latter's ID. The root group is retrieved as the **IExtranetGroupHierarchyNode** object, which allows you to further iterate its sub-groups and query or use their specific properties. It requires the extranet provider's name.

```
public static IExtranetGroupHierarchyNode GetRootGroup(string providerName,
Guid groupId);
```

Parameters

providerName	The name of the Extranet provider.
groupId	The group's ID.

Return Value

The **IExtranetGroupHierarchyNode** object if the operation succeeds; otherwise, null.

3.2.6 GetGroupHierarchy (1)

This method retrieves an existing group as the **IExtranetGroupHierarchyNode** object by its group ID, which allows you to further iterate its sub-groups and query or use their specific properties. It requires the extranet provider's name.

```
public static IExtranetGroupHierarchyNode GetGroupHierarchy(string
providerName, Guid groupId);
```

Parameters

providerName	The name of the Extranet provider.
groupId	The group's ID.

Return Value

The **IExtranetGroupHierarchyNode** object if the operation succeeds; otherwise, null.

3.2.7 GetGroupHierarchy (2)

This method retrieves all the existing groups as an enumerable collection of **IExtranetGroupHierarchyNode** objects (hierarchy), which allows you to further iterate its sub-groups and query or use their specific properties. It requires the extranet provider's name.

```
public static IEnumerable<IExtranetGroupHierarchyNode>
GetGroupHierarchy(string providerName);
```

Parameters

providerName	The name of the Extranet provider.

Return Value

An enumerable collection of **IExtranetGroupHierarchyNode** objects if the operation succeeds; otherwise, null.

3.2.8 GetGroupSelectOptions

This method retrieves all the groups that can be assigned to web pages and media folders and files as an enumerable collection of key/value pairs with group IDs and group names. The values can be further used, for example, in the selector widgets.

You can configure whether the group name should be a "short" name or a "long" name. The long name includes the short name (the name of the group) and the path to the group from the root group in the group hierarchy.

You can also indicate whether the root group should be retrieved as well.

```
public static IEnumerable<KeyValuePair<Guid, string>>
GetGroupSelectOptions(string providerName, bool allGroups, bool longNames);
```

Parameters

providerName	The name of the Extranet provider.
allGroups	This parameter indicates whether the root group should be retrieved along with all the sub-groups.
longNames	This parameter indicates whether the group names should include the hierarchical “paths” (short names vs. long names)

Return Value

An enumerable collection of key/value pairs with group IDs and group names if the operation succeeds; otherwise, null.

3.2.9 GroupLongName

This method gets a “long” name of an existing group by its ID as a path that includes names of all the parent groups separated by the specified delimiter. It requires the extranet provider’s name.

```
public static string GroupLongName(string providerName, Guid groupId,
string delimiter);
```

Parameters

providerName	The name of the Extranet provider.
groupId	The group’s ID.
delimiter	A character used for separating groups in the path (e.g. “/”)

Return Value

A long name of the group, which includes the names of its parent groups in the group hierarchy.

3.2.10 UpdateGroup

This method updates properties of an existing group in the extranet identified by its provider’s name. Before updating the group, you should get the existing group as an **IExtranetGroup** object and modify the desired properties with new values.

```
public static void UpdateGroup(string providerName, IExtranetGroup group);
```

Parameters

providerName	The name of the Extranet provider.
group	An **IExtranetGroup** object.

3.2.11 DeleteGroup

This method deletes an existing group in the extranet identified by its provider’s name. The group is identified by its group ID.

```
public static void DeleteGroup(string providerName, Guid groupId);
```

Parameters

providerName	The name of the Extranet provider.
groupId	The group's ID.

3.2.12 ValidateUserGroupsToItemGroups

This method validates the user groups as a collection of their group IDs against the extranet-protected data items (such as web page or media files) as a collection of their item IDs in the extranet identified by its provider's name.

The validation result is used to determine whether the user as a member of the specified groups has access to the specified items.

The method can be used not only with web pages and media files but also with any custom items that can be protected with the extranet.

```
public static bool ValidateUserGroupsToItemGroups(string providerName,
IEnumerable<Guid> userGroups, IEnumerable<Guid> itemGroups);
```

Parameters

providerName	The current extranet provider's name
userGroups	An enumerable collection of group IDs
itemGroups	An enumerable collection of IDs of items such as pages, media files or custom data items

Return Value

If the validation is successful, it returns true; otherwise, false.

3.3 Managing Relations between Users and Groups

In the Extranet add-on the user can be a member of multiple groups. The opposite is also true - the group can have multiple users. Using the **ExtranetFacade** methods, you can assign a user to a number of groups and a group to a number of users. Besides, you can get a list of users in a specific group as well as a list of groups of which the user is a member.

3.3.1 GetUsersFromGroup

This method gets user accounts as an enumerable collection of **IExtranetUser** objects, assigned to a group identified by its group ID. You can further iterate this collection and query or use the specific user's properties. It requires the extranet provider's name.

```
public static IEnumerable<IExtranetUser> GetUsersFromGroup(string
providerName, Guid groupId);
```

Parameters

providerName	The name of the Extranet provider.
groupId	The group's ID.

Return Value

A collection of **IExtranetUser** objects if the operation succeeds; otherwise, null.

3.3.2 GetGroupIdListForUser

This method gets groups as an enumerable collection of group IDs, to which a user identified by its user ID is assigned. You can further iterate the collection and use these group IDs. It requires the extranet provider's name.

```
public static IEnumerable<Guid> GetGroupIdListForUser(string providerName,
Guid userId);
```

Parameters

providerName	The name of the Extranet provider.
userId	The user's ID.

Return Value

A collection of group IDs (GUIDs) if the operation succeeds; otherwise, null.

3.3.3 SetGroupsForUsers

This method assigns a collection of groups identified by their group IDs to a user identified by its user ID. You should first build an enumerable collection of existing groups and then pass it as one of the method's input parameters. The method also requires the extranet provider's name.

```
public static void SetGroupsForUser(string providerName, Guid userId,
IEnumerable<Guid> groupIdList);
```

Parameters

providerName	The name of the Extranet provider
userId	The user's ID
groupIdList	A collection of group IDs

3.3.4 SetUsersForGroups

This method assigns a collection of users identified by their user IDs to a group identified by its group ID. You should first build an enumerable collection of existing users and then pass it as one of the method's input parameters. The method also requires the extranet provider's name.

```
public static void SetUsersForGroup(string providerName, Guid groupId,
IEnumerable<Guid> userIdList);
```

Parameters

providerName	The name of the Extranet provider.
groupId	The group's ID.
userIdList	A collection of user IDs

3.4 Managing Extranet Security on Websites

The usual scenario for managing extranet security on a website is as follows:

1. The extranet security is assigned to a website (the root, or home, page). This includes specifying the extranet provider and the page that will be used for user to log in to the extranet (a “Login” page).
2. One or more extranet groups are assigned to pages (which automatically includes all its sub-pages) to control what users in these groups can or cannot access on the website.

In the first case, you will need to initialize or use an IPageExtranet object, which includes the required information on the provider, login page and the root page ID.

In the second case, you will create or use an IPageSecurity object that holds necessary information about the extranet, page and security implemented by assigning a specific group to the page.

A number of methods are provided in the **ExtranetFacade** for managing the extranet security on websites.

3.4.1 IPageExtranet

IPageExtranet extends **IData** with properties that hold information about the extranet settings for a website (identified by its root page ID), which include the extranet provider’s name and the Login page ID.

```
public interface IPageExtranet : IData
{
    Guid LoginPageId { get; set; }
    Guid PageId { get; set; }
    string ProviderName { get; set; }
}
```

Properties

PageId	The website’s root page ID
ProviderName	The extranet provider’s name
LoginPageID	The ID of the page used to log in to the extranet

3.4.2 IPageSecurity

IPageSecurity extends **IData** with properties that hold information about the extranet security settings for a web page (identified by its page ID), which include the ID of the extranet group assigned to this page as well as the extranet ID (root page ID) and its provider’s name.

```
public interface IPageSecurity : IData
{
       Guid Id { get; set; }
       Guid ExtranetId { get; set; }
       string ProviderName { get; set; }
       Guid PageId { get; set; }
       Guid GroupId { get; set; }
}
```

Properties

Id	The page security ID
ExtranetId	The extranet ID, which is the extranet-protected website's root page ID
ProviderName	The extranet provider's name
PageId	The page ID
GroupID	The ID of the extranet group assigned to the page

3.4.3 AddNewExtranetToPage

This method adds the extranet settings to the website identified by its root page ID. The extranet settings include the extranet provider's name and ID of the page used to log in to the extranet.

```
public static void AddNewExtranetToPage(Guid pageId, string providerName,
Guid selectedLoginPage);
```

Parameters

pageId	The website's root page ID
providerName	The extranet provider's name
selectedLoginPage	The ID of the page used to log in to the extranet

3.4.4 GetExtranetToPage (1)

This method gets the extranet settings as an **IPageExtranet** object on a website identified by its root page ID.

```
public static IPageExtranet GetExtranetToPage(Guid pageId);
```

Parameters

pageId	The website's root page ID

Return Value

An **IPageExtranet** object if the ID is that of the website's root page; otherwise, null.

3.4.5 GetExtranetToPage (2)

This method gets the extranet settings of all the websites as a queryable collection of **IPageExtranet** objects.

```
public static IQueryable<IPageExtranet> GetExtranetToPage();
```

Return Value

A queryable collection of **IPageExtranet** objects if one or more websites are extranet-protected; otherwise, an empty collection.

3.4.6 GetExtranetToPageByLoginPageId

This method gets the extranet settings as an **IPageExtranet** object by its Login page ID.

```
public static IPageExtranet GetExtranetToPageByLoginPageId(Guid pageId);
```

Parameters

pageId	The ID of the page used for logins to the extranet

Return Value

An **IPageExtranet** object if the ID is that of the Login page used in the extranet; otherwise, null.

3.4.7 GetCurrentExtranet

This method gets current extranet settings as an **IPageExtranet** object in the context of the current website. You can further query the object for the provider's name, the website's root page ID and Login page ID.

```
public static IPageExtranet GetCurrentExtranet();
```

Return Value

An **IPageExtranet** object if extranet set up on the current website; otherwise, null.

3.4.8 GetRootPageInExtranet

This method gets the extranet settings as an IPageExtranet object for a specific page on website identified by its page ID.

```
public static IPageExtranet GetRootPageInExtranet(Guid pageId);
```

Parameters

pageId	The page ID

Return Value

An **IPageExtranet** object if the page is on an extranet-protected website; otherwise, null.

3.4.9 GetCurrentLoginPage

This method gets the ID of the Login page used in the extranet on the current website.

```
public static Guid GetCurrentLoginPage();
```

Return Value

The Login page ID.

3.4.10 UpdateExtranetToPage

This method updates extranet settings of a page. Before calling this method, you should initialize **IPageExtranet** object with desired values and then pass it as an input parameter.

```
public static void UpdateExtranetToPage(IPageExtranet extranet);
```

Parameters

extranet	An **IPageExtranet** object

3.4.11 DeleteExtranetToPage

This method deletes extranet settings from the website identified by its root page ID.

```
public static void DeleteExtranetToPage(Guid pageId);
```

Parameters

pageId	The page's ID

3.4.12 GetPageSecurityPages

This method gets all the pages protected by an extranet group identified by its group ID in the extranet identified by its provider's name. The pages are returned as an enumerable collection of **IPageSecurity** objects.

```
public static IEnumerable<IPageSecurity> GetPageSecurityPages(string
providerName, Guid groupId);
```

Parameters

providerName	The extranet provider's name
groupID	The extranet group's ID

Return Value

An enumerable collection of **IPageSecurity** objects if the group is assigned to any page; otherwise, an empty collection.

3.4.13 SetPageSecurityGroups

This method assigns a list of extranet groups as a collection of group IDs to the page identified by its page ID. The extranet should be identified by its provider's name.

```
public static void SetPageSecurityGroups(string providerName, Guid
extranetId, Guid pageId, IEnumerable<Guid> groupIdList);
```

Parameters

providerName	The extranet provider's name
extranetId	The ID of the root page on a website
pageId	The page ID
groupIdList	An enumerable collection of group IDs to be assigned to the page

3.4.14 GetPageSecurityGroups

This method retrieves the groups as an enumerable collection of group IDs from a page by its page ID in the extranet identified by its provider's name.

```
public static IEnumerable<Guid> GetPageSecurityGroups(string providerName,
Guid pageId);
```

Parameters

providerName	The extranet provider's name
pageId	The page's ID

Return Value

An enumerable collection of group IDs if any groups are assigned to the page; otherwise, an empty collection.

3.4.15 GetPageSecurityInheritedGroups

This method retrieves the groups inherited from all the parent pages as an enumerable collection of group IDs for a page identified by its page ID in the extranet identified by its provider's name.

```
public static IEnumerable<Guid> GetPageSecurityInheritedGroups(string
providerName, Guid pageId);
```

Parameters

providerName	The extranet provider's name
pageId	The page ID

Return Value

An enumerable collection of group IDs if any groups are inherited on the page; otherwise, an empty collection.

3.4.16 DeleteGroupFromPageSecurity

This method deletes an existing group identified by its group ID from all the pages protected by the extranet identified by its provider's name.

```
public static void DeleteGroupFromPageSecurity(string providerName, Guid
groupId);
```

Parameters

providerName	The extranet provider's name
groupId	The group ID

3.4.17 GetExtranetSitemapXml

This method gets the XML-formatted site map in the context of the current extranet for the groups identified by their group IDs.

You can configure the sitemap by indicating whether to exclude the restricted pages from the sitemap as well as whether to include additional attributes for each active page (active

page annotations) and attributes that will list the groups assigned to each page (as group IDs).

```
public static IEnumerable<XElement> GetExtranetSitemapXml(IEnumerable<Guid>
groups, bool removeBlockedPages, bool addGroups, bool
activePageAnnotations);
```

Parameters

groups	An enumerable collection of group IDs
removeBlockedPages	The parameter that indicates whether to exclude the restricted pages from the sitemap
addGroups	The parameter that indicates whether to include attributes that will list the groups assigned to each page (as group IDs).
activePageAnnotations	The parameter that indicates whether to include additional attributes for each active page (e.g. “isopen”, “iscurrent”) in the sitemap

Return Value

An XML-formatted site map.

3.5 Managing Extranet Security on Media

The extranet security can be applied not only to websites but also to media folders and, thus, media files. The methods used on media are similar to those used on websites and web pages.

The usual scenario for managing extranet security on media is as follows:

1. The extranet security is assigned to a media folder. This includes specifying an extranet provider and the ID of the website's root page (extranet ID) as well as the path to the media folder to protect.
2. One or more extranet groups are assigned to media folders or files (which automatically includes all its sub-folders and files) to control what users in these groups can or cannot access.

In the first case, you will need to initialize or use an IMediaExtranet object, which includes the required information on the provider, the extranet-protected website and the path to a media in the Media Archive.

In the second case, you will create or use an IMediaSecurity object that holds necessary information about the extranet provider, extranet-protected website, media file or folder and security implemented by assigning a specific group to the media.

A number of methods are provided in the **ExtranetFacade** for managing the extranet security on media.

3.5.1 IMediaExtranet

IMediaExtranet extends **IData** with properties that hold information about the extranet settings for media (identified by its media path), which include the extranet provider's name and the extranet ID and the path to the media.

```
public interface IMediaExtranet : IData
{
    string CompositePath { get; set; }
    Guid ExtranetId { get; set; }
    string ProviderName { get; set; }
}
```

Properties

CompositePath	The path to a media folder in the Media Archive
ExtranetID	The ID of the root page on an extranet-protected website
ProviderName	The extranet provider's name

3.5.2 IMediaSecurity

IMediaSecurity extends **IData** with properties that hold information about the extranet security settings for a media (identified by its path), which include the ID of the extranet group assigned to this page as well as the extranet ID and its provider's name.

```
public interface IMediaSecurity : IData
{
	Guid Id { get; set; }
	Guid ExtranetId { get; set; }
	string ProviderName { get; set; }
	string CompositePath{ get; set; }
	Guid GroupId { get; set; }
}
```

Properties

Id	The media security ID
ExtranetId	The ID of the root page on an extranet-protected website
ProviderName	The extranet provider's name
CompositePath	The path to a media file or folder in the Media Archive
GroupID	The ID of the extranet group assigned to the media

3.5.3 AddNewExtranetToMedia

This method adds the extranet settings to a media folder at the specified path. The extranet settings include the extranet provider's name and the ID of the root page on an extranet-protected website.

```
public static void AddNewExtranetToMedia(string compositePath, string
providerName, Guid ExtranetId);
```

Parameters

compositePath	The path to a media folder
providerName	The extranet provider's name
extranetID	The ID of the root page on an extranet-protected website

3.5.4 GetExtranetToMedia

This method gets all the extranet settings of all media folders in the media archive as a queryable collection of **IMediaExtranet** objects.

```
public static IQueryable<IMediaExtranet> GetExtranetToMedia();
```

Return Value

A queryable collection of **IMediaExtranet** objects if any folder is extranet-protected; otherwise, an empty collection.

3.5.5 GetExtranetToMedia

This method gets the extranet settings as an **IMediaExtranet** object from a media folder at the specified path.

```
public static IMediaExtranet GetExtranetToMedia(string compositePath);
```

Parameters

compositePath	The path to a media folder

Return Value

An **IMediaExtranet** object if the media folder is extranet-protected; otherwise, null.

3.5.6 UpdateExtranetToMedia (1)

This method updates extranet settings of a media folder. Before calling this method, you should create and initialize **IMediaExtranet** object and then pass it as an input parameter.

```
public static void UpdateExtranetToMedia(IMediaExtranet extranet);
```

Parameters

extranet	An **IPageExtranet** object

3.5.7 DeleteExtranetToMedia (2)

This method deletes extranet settings from a media folder at the specified path.

```
public static void DeleteExtranetToMedia(string compositePath);
```

Parameters

compositePath	The path to a media folder

3.5.8 GetRootMediaInExtranet

This method gets the root extranet-protected media folder as an **IMediaExtranet** object for a media at the specified path.

```
public static IMediaExtranet GetRootMediaInExtranet(string compositePath);
```

Parameters

compositePath	The path to a media

Return Value

An **IMediaExtranet** object, if the media file or folder is located in the extranet-protected folder; otherwise, null.

3.5.9 GetSubMediaInExtranet

This method detects whether the media folder at the specified path contains an extranet-protected area in the Media Archive.

```
public static IMediaExtranet GetSubMediaInExtranet(string compositePath);
```

Parameters

compositePath	The path to a media

Return Value

An **IMediaExtranet** object, if the media folder contains an extranet-protected media area; otherwise, null.

3.5.10 GetMediaSecurityMedia

This method gets all the media folders protected by an extranet group identified by its group ID in the extranet identified by its provider's name. The media folders are returned as an enumerable collection of **IMediaSecurity** objects.

```
public static IEnumerable<IMediaSecurity> GetMediaSecurityMedia(string
providerName, Guid groupId);
```

Parameters

providerName	The extranet provider's name
groupID	The extranet group ID

Return Value

An enumerable collection of **IMediaSecurity** objects if at least one media folder is extranet protected; otherwise, an empty collection.

3.5.11 GetMediaSecurityGroups

This method retrieves the groups as an enumerable collection of group IDs from a media by its path in the extranet identified by its provider's name.

```
public static IEnumerable<Guid> GetMediaSecurityGroups(string providerName,
string CompositePath);
```

Parameters

providerName	The extranet provider's name
CompositePath	The path to a media folder or file

Return Value

An enumerable collection of group IDs if any media is protected by any group; otherwise, an empty collection.

3.5.12 GetMediaSecurityInheritedGroups

This method retrieves the groups inherited from parent media folders as an enumerable collection of group IDs from a media by its path in the extranet identified by its provider's name.

```
public static IEnumerable<Guid> GetMediaSecurityInheritedGroups(string
providerName, string CompositePath);
```

Parameters

providerName	The extranet provider's name
CompositePath	The path to a media folder

Return Value

An enumerable collection of group IDs if any media is protected by any inherited group; otherwise, an empty collection.

3.5.13 SetMediaSecurityGroups

This method assigns a list of extranet groups as group IDs to the media identified by its media ID. The extranet should be identified by its provider's name and the ID of the root page on an extranet-protected website.

```
public static void SetMediaSecurityGroups(string providerName, Guid
extranetId, string mediaId, IEnumerable<Guid> groupIdList);
```

Parameters

providerName	The extranet provider's name
extranetId	The ID of the extranet
mediaId	The media ID
groupIdList	An enumerable collection of group IDs to be assigned to the media

3.5.14 DeleteGroupFromMediaSecurity

This method deletes an existing group by its group ID from all the media folders and files in the Media Archive.

```
public static void DeleteGroupFromMediaSecurity(string providerName, Guid
groupId);
```

Parameters

providerName	The extranet provider's name
groupId	The group ID

3.5.15 GetExtranetMediaArchiveSitemapXml

This method gets the XML-formatted site map in the context of the current extranet for the groups identified by their group IDs.

You can configure the sitemap by indicating whether to exclude the restricted media from the sitemap as well as whether to include attributes that will list the groups assigned to each media (as group IDs).

```
public static IEnumerable<XElement>
GetExtranetMediaArchiveSitemapXml(IEnumerable<Guid> userGroups, bool
removeBlockedMedia, bool addGroups);
```

Parameters

userGroups	An enumerable collection of group IDs
removeBlockedMedia	The parameter that indicates whether to exclude the restricted media from the site map
addGroups	The parameter that indicates whether to include attributes that will list the groups assigned to each media (as group IDs).

Return Value

An XML-formatted site map.

3.6 Managing Extranet Settings

Each extranet provider has a number of properties that can be read and used in the code by some of the **ExtranetFacade** methods.

3.6.1 IExtranetProviderSettings

IExtranetProviderSettings provides information about the extranet provider, which includes its name and title and indictates whether user accounts and user groups can be updated, passwords read, or the Forms Authentication supported.

All the properties are read-only.

```
public interface IExtranetProviderSettings
{
    bool FormsLoginSupported { get; }
    bool GroupsUpdatable { get; }
    string Name { get; }
    string Title { get; }
    bool UserPasswordsReadable { get; }
    bool UsersUpdatable { get; }
}
```

Properties

Name	The name of the extranet provider by which the latter is referred to in the code (read-only)
Title	The title of the extranet provider by which the latter is visible to end users in the GUI (read-only)
FormsLoginSupported	The property that indicates whether the Forms Authentication is supported (read-only)
UsersUpdatable	The property that indicates whether changes in a user account's settings are allowed (read-only)
GroupsUpdatable	The property that indicates whether changes in a user group's settings are allowed (read-only)
UserPasswordsReadable	The property that indicates whether reading user passwords is allowed (read-only)

3.6.2 GetCurrentProviderName

This method returns the current extranet provider's name. The name is used internally in the code.

```
public static string GetCurrentProviderName();
```

Return Value

A string that specifies the current extranet provider's name.

3.6.3 GetCurrentProviderTitle

This method returns the current extranet provider's title. The title is used externally in the GUI.

```
public static string GetCurrentProviderTitle();
```

Return Value

A string that specifies the current extranet provider's title.

3.6.4 GetDefaultProviderName

This method returns the default extranet provider's name. The name is used internally in the code.

```
public static string GetDefaultProviderName();
```

Return Value

A string that specifies the default extranet provider's name.

3.6.5 GetProviderSettings (1)

This method retrieves the settings for all the extranet providers as an enumerable collection of **IExtranetProviderSettings** objects.

```
public static IEnumerable<IExtranetProviderSettings> GetProviderSettings();
```

Return Value

An enumerable collection of **IExtranetProviderSettings** objects.

3.6.6 GetProviderSettings (2)

This method retrieves the settings as an **IExtranetProviderSettings** object for the extranet identified by its provider's name.

```
public static IExtranetProviderSettings GetProviderSettings(string
providerName);
```

Parameters

providerName	The name of the extranet provider whose settings are retrieved.

Return Value

An **IExtranetProviderSettings** object that holds the provider's settings.

3.7 Managing Logins and Validation

A number of methods are used for the front-end operations such as logging in and out as well as validating users and groups.

3.7.1 Login

This method logs the user with the specified username and password in to the extranet. You can specify whether to keep the user logged in ("remember") between sessions and whether to refresh the page the user has come from to update the login status.

```
public static bool Login(string userName, string password, bool rememberMe,
bool refreshOnLogin);
```

Parameters

userName	The name of the user to log in with to the extranet
password	The password of the user to use when logging in to the extranet
rememberMe	This parameter indicates whether to keep the user logged in between sessions
refreshOnLogin	This parameter indicates whether to refresh the page the user has come from to update the login status

Return Value

It the user logs in to the extranet, it returns true; otherwise, false.

3.7.2 Logout

This method logs the current user out of the extranet.

```
public static void Logout();
```

3.7.3 IsAuthenticated

This method indicates whether the user in the current context is authenticated on the extranet.

```
public static bool IsAuthenticated();
```

Return Value

If the user is authenticated, it returns true; otherwise, false.

3.7.4 ValidateUser

This method validates the user's credentials such as the username and password against the user accounts in the extranet identified by the provider's name. If the validation succeeds, the method returns the user's ID in its corresponding output parameter, which can be further used in the code to refer to the user.

```
public static bool ValidateUser(string providerName, string login, string
password, out Guid userId);
```

Parameters

providerName	The current extranet provider's name
login	The name of the user to log in with to the extranet
password	The password of the user to use when logging in to the extranet
userId	[out] The user's ID in the extranet returned if the validation is successful.

Return Value

If the validation is successful, it returns true; otherwise, false.

4 About Extranet Security

The Extranet add-on protects pages and media files from being requested by website visitors who are not authenticated or authorized.

When restricting access to a page, you are limiting the group of people who can see that page (and all its subpages) to the people who have a valid login and belong to at least one Extranet group has the access.

The same is the case for media files.

4.1 How it works

C1 CMS has exactly two public endpoints that can serve pages and media files. These endpoints are responsible for converting an HTTP request into API calls that deliver the requested resource.

Before these API calls are executed, the endpoints will validate requests with the Extranet add-on and abort the request if not properly validated. Aborted requests are redirected to the website login page. If properly validated, the request is allowed, but any public caching is explicitly denied.

The technical mechanism used to enable this request validation is described here: "How can I validate users before a page or media file is being served?"

4.2 What is protected?

The validation layer is inserted between the public endpoints and the API constructing the response. This means that the Extranet add-on is protecting the public endpoints and ensures that pages and media are only handed to the users who have been authorized by the extranet administrator.

4.3 When are pages or media not protected?

The data stores of C1 CMS are not 'encapsulated' by this protection and any direct Data API calls will be allowed by C1 CMS. This means that developers who write their own public endpoints serving pages or media files must take the "RenderingResponseHandler" security feature into account to maintain the security level provided by the Extranet add-on.

4.4 What about administrative users?

Users with administrative access to the CMS Console can work with pages and media files according to their CMS Console security settings. These settings can be limited so that only specific users are allowed to browse and manipulate specific page sections and media folders. Browsing and previewing protected pages require a valid extranet login.

4.5 Secure communication

By default the Extranet add-on will use HTML form based login validation where users submit their username and password in plain text.

You can ensure that this communication along with page browsing and file downloads are encrypted by installing a SSL (Secure Socket Layer) certificate on the website. This will ensure that traffic can be transmitted in a secure way via HTTPS (Secure HTTP).

CMS Console users, which manage extranet user accounts, edit or upload extranet-protected content or files, can use HTTPS for these purposes. This will ensure that all password, data and file communication between the CMS Console user and the server is maintained in a secure and encrypted manner.

Some websites might require that sensitive data and files are stored in an encrypted state on the server. Neither the Extranet add-on nor C1 CMS provides storage encryption services by default. However, such features can be developed as add-ons to C1 CMS and should work with this extranet “as is”.

5 Extranet Functions

The Extranet add-on comes with a number of CMS functions.

Some of these functions handle UI-specific tasks on your website, for example, displaying a login form (Composite.Community.Extranet.LoginForm), a signup form (Composite.Community.Extranet.SignupForm) or the login status (Composite.Community.Extranet.LoginStatus) on pages.

Some other functions (for example, Composite.Community.Extranet.User.IsAuthenticated) work more like building blocks you can use in your existing functionality on the website, for example, in Razor or XSLT functions.

You can read about all these functions by generating their documentation in the CMS Console:

1. In the **Functions** perspective, expand **All Functions** | **Composite** | **Community**.
2. Select **Extranet**.
3. Click **Generate Documentation** on the toolbar.
4. Locate these functions and read their descriptions.

5.1 ExtranetSitemapXml

To generate a sitemap on the website, you normally use Composite.Pages.SitemapXml in your XSLT - either directly or indirectly (in other functions that use it).

If your website has extranet-protected pages, it makes sense to generate a sitemap that only shows links to the pages the currently logged-in extranet user has access to.

In this case, you should use the Composite.Community.Navigation.ExtranetSitemapXml function:

- This function excludes from the resulting sitemap XML all the protected pages the current extranet user has no access to, based on the extranet groups he or she is a member of.
- It also allows you to arbitrarily include some excluded protected pages by specifying additional group or groups (via the user) assigned to these pages.
- If you choose to, you can have the function generate additional info for pages - extranet groups they are associated with and whether they are currently active ("is open, "is current").

The function has the following parameters:

- **ProviderName** (String): The name of the extranet provider the website is set to. "Extranet" by default, which is the provider the Extranet add-on comes with. If you have more than one extranet provider on your website, you should explicitly select the proper name. The provider name is added to the homepage in the "providerName" attribute.
- **UserId** (Guid): The ID of an extranet user the protected pages he/she is assigned to (via extarnet groups) must be additionally included in the resulting sitemap XML. No default value.
- **GroupId** (Guid): The ID of an extranet group the protected pages it is assigned to must be additionally included in the resulting sitemap XML. No default value.
- **RemoveBlockedPages** (Boolean): If 'True', all the protected pages will be removed from the resulting sitemap XML (except for the pages the current extranet user has access to and/or included via UserId and GroupId). 'True' by default.
- **AddGroups** (Boolean): If 'True', the page info will include attributes that will list the groups (as group IDs) assigned to each page ("extranetGroups", "isInherited"). 'False' by default.
- **ActivePageAnnotations** (Boolean): If 'True', the page info will include attributes for each active page (e.g. "isopen", "iscurrent") in the sitemap. 'True' by default.

(Some of these parameters have the same purpose as the API function GetExtranetSitemapXml.)

The Composite.Community.Navigation.ExtranetSitemapXml function generates a hierarchy of "Page" elements. Each "Page" element has a few extranet-related attributes (along with the standard Sitemap attributes).

The "isBlocked" attribute set to 'True' or 'False' based on whether the page is protected.

The homepage has the "providerName" attribute set to the name of the extranet provider (default or specific).

If the “AddGroups” parameter is set to 'True', two more attributes added to "Page" elements:

- The "isInherited" attribute is set to 'True' if the protection is inherited from the parent page.
- The "extranetGroups" attribute lists comma-separated IDs of extranet groups assigned to this page.

```
<Page Id="a3055286-0e90-4b04-99dd-fb1a61dde0bf" Title="Frontpage"
MenuTitle="Frontpage" UrlTitle="Home" Description="" ChangedDate="2011-07-
20T13:47:38.9133156+03:00" ChangedBy="admin" URL="/Home" Depth="1"
isopen="true" iscurrent="true" providerName="Default" isBlocked="false"
xmlns="">
  <Page Id="465e8e08-6068-4a65-b313-934324d7c6a8" Title="2 Columns"
MenuTitle="2 Columns" UrlTitle="2Columns" Description="" ChangedDate="2011-
07-20T12:48:33.3725001+03:00" ChangedBy="admin" URL="/Home/2Columns"
Depth="2" isBlocked="true" isInherited="false" extranetGroups="5880857b-
1fc0-46b6-8de8-b4c5bb730826"/>
  <Page Id="b6a13c6c-2b13-4df7-bdcf-282dbbba3635" Title="3 Columns"
MenuTitle="3 Columns" UrlTitle="3-Columns" Description=""
ChangedDate="2011-07-20T12:48:33.5135081+03:00" ChangedBy="admin"
URL="/Home/3-Columns" Depth="2" isBlocked="false" />
</Page>
```