Table of Contents

1 Stimulsoft Reports Server API SDK

1.1 Users

Users

To access information about users of the Stimulsoft Reports Server, there is a specialized user class called `StiUserConnection` that automatically handles much of the functionality required for user account management.
With this class, you will be able to add user account functionality in your app.

In order to use the class `StiUserConnection` you should create an instance of `StiServerConnection` and call one of its methods (`StiServerConnection.Users`).

Signing Up

To create a new user, you must use the method `SignUp()` or asynchronous version `SignUpAsync()`:

```
public void SignUp()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.SignUp("UserName", "Password");
}
```

Or asynchronous method:

```
public async void SignUpAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.SignUpAsync("UserName", "Password");
}
```

This call will create a new user in your instance of Stimulsoft Reports Server. Before it does this, it also checks to make sure that both the username and email are unique. Enter the password – on the server side it is stored as a hash and is never sent to the client as plaintext.

Also we have two overridden methods for signup action with additional user info (first name and last name):

```
public void Signup()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.SignUp("UserName@example.com", "Password", "FirstName",
"LastName");
}

public async void SignupAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.SignUpAsync("UserName@example.com", "Password",
"FirstName", "LastName");
}
```

You must to use an email address as the username.

Logging In

Log in registered user uses the class method `Login()` (`LoginAsync()`).

```
public void LogIn()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");
}
```

Or asynchronous method:

```
public async void LogInAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");
}
```

Getting User Info

In order to obtain information about the user, use the methods `GetByName()` or `GetByKey()` that return an object `StiUser`. This object contains full information about the user (key of user role, workspace and root folder, OAuth identifier and type of authorization method, first name, last name, user name (email), picture of avatar, some flags (enabled, activated) and times when this user is created, last modified and last logged).
Before calling the method `GetByName()` or `GetByKey()` you must log in as a user whose rights are allowed to have access to the necessary information.

```
public void GetUserInfo()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var userJohn = connection.Users.GetByName("John@example.com");
    var johnLastName = userJohn.LastName;
    var johnLogin = userJohn.LastLogin;

    var userScott = connection.Users.GetByKey("ScottKey");
    var scottLastName = userScott.LastName;
    var scottLogin = userScott.LastLogin;
}
```

Or asynchronous method:

```
public async void GetUserInfoAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var userJohn = await connection.Users.GetByNameAsync("John@example.com");
    var johnLastName = userJohn.LastName;
    var johnLogin = userJohn.LastLogin;

    var userScott = await connection.Users.GetByKeyAsync("ScottKey");
    var scottLastName = userScott.LastName;
    var scottLogin = userScott.LastLogin;
}
```

Getting a list of Users

To find or process information about users of the system, there is a method that allows you to get a list of all objects `StiUser`, to which the current user can access. Use the method `FetchAll()` (`FetchAllAsync()`).

```
public void ProcessUsersInfo()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");
    var users = connection.Users.FetchAll();

    //find user with last name "Smith"
    var mrSmith = users.First(a => a.FirstName == "Smith");
}
```

Asynchronous method:

```
public async void ProcessUsersInfoAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");
```

```
    var users = await connection.Users.FetchAllAsync();

    //is exist user with name "John"
    var isJohnExists = users.Any(a => a.LastName == "John");
}
```

Changing User info

Data stored in a `StiUser` object can be modified. The possibility of change is determined by the level of access the user on whose behalf the operations are performed.

```
public void ChangeUserLastName()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var userJohn = connection.Users.GetByName("John@example.com");
    userJohn.LastName = "Smith";
    userJohn.Save();
}
```

Asynchronous method:

```
public async void ChangeUserLastNameAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var userJohn = await connection.Users.GetByNameAsync("John@example.com");
    userJohn.LastName = "Smith";
    await userJohn.SaveAsync();
}
```

Changing User password

Passwords are stored in the system in the form of hashes and object `StiUser` not return hem. To change the password used method `ChangePassword()` (`ChangePasswordAsync()`). The parameters you must specify the current password and new.

```
public void ChangeUserPassword()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var userJohn = connection.Users.GetByName("John@example.com");
    userJohn.ChangePassword("JohnPassword", "NewPassword");
}
```

Asynchronous method:

```
public async void ChangeUserPasswordAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var userJohn = await connection.Users.GetByNameAsync("John@example.com");
    await userJohn.ChangePasswordAsync("JohnPassword", "NewPassword");
}
```

Creating new User

Creating new users is through the creation of a new object `StiUser` and its storing through methods `Save()` or `SaveAsync()`.

```
public void CreateNewUser()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var newUser = connection.Users.New();
    newUser.UserName = "UserName@example.com";
    newUser.Password = "UserPassword";
    newUser.Save();
}
```

Asynchronous method:

```
public async void CreateNewUserAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var newUser = connection.Users.New();
    newUser.UserName = "UserName@example.com";
    newUser.Password = "UserPassword";
    await newUser.SaveAsync();
}
```

Deleting User

Remove element by calling `Delete()` (`DeleteAsync()`) the object class `StiUser` or by calling one of the methods `DeleteByKey()`, `DeleteByKeyAsync()`, `DeleteByName()`, `DeleteByNameAsync()` from collection `StiServerConnection.Users`:

```
public void DeleteItem()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var user = connection.Users.GetByKey("UserKey");
    if (user != null)
    {
        user.Delete();
    }

    connection.Users.Logout();
}
```

Asynchronous example:

```
public async void DeleteItemAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    await connection.Users.DeleteByNameAsync("JohnSmith");

    await connection.Users.LogoutAsync();
}
```

Logging Out

After all actions the user must log out. To do this, you can use the class methods `Logout()` or `LogoutAsync()`.

```
public void Logout()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");
    connection.Users.Logout();
}
```

Asynchronous method:

```
public async void LogoutAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");
    await connection.Users.LogoutAsync();
}
```

1.2 Items

Items

The basis of the functional Stimulsoft Reports Server are operations on items. The item is an object that can be visually observed in the object tree on the left side of the interface of the client application.
All items are inherited from the root abstract class `StiItem` and, depending on the functionality provided, are one of the following classes:
`StiCalendarItem` (used to create a scheduler)
`StiContactListItem` (used for sending reports via e-mail)
`StiDataSourceItem` (used to connect external data sources)
`StiFileItem` (used to connect external data sources)
`StiFolderItem` (provides a hierarchical tree structure elements)
`StiReportSnapshotItem` (rendered data report)
`StiReportTemplateItem` (report template for building)
`StiSchedulerItem` (used to automate actions)

To work with elements uses a unique identifier - keys. They are assigned automatically when you create elements. Get the key can be from the Navigator (in the context menu item by selecting Access Key).

Creating and saving Items

Position in the hierarchical tree structure of elements defined by the connection element and folders. Therefore, to create the item, you must first create an object of type `StiFolderItem`, specifying its position in the tree, and then use one of the methods to create an item of a particular type. The root element of the tree is represented by an instance of the class `StiFolderItem: StiServerConnection.Items.Root`.
After defining properties of the new item is necessary to perform his method `StiItem.Save ()` or `StiItem.SaveAsync ()`.
The following example shows how to create a folder in the root of the tree of elements and add the calendar (describes Monday) to this folder:

```
public void CreateNewCalendarItem()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var folderItem = connection.Items.Root.NewFolder("folder");
    folderItem.Save();

    var calendarItem = folderItem.NewCalendar("NewCalendar");
    calendarItem.Dates.Add(new StiCalendarDate("Monday", StiDaysOfWeek.Monday));
```

```
    calendarItem.Save();
}
```

Asynchronous method:

```
public async void CreateNewCalendarItemAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var folderItem = connection.Items.Root.NewFolder("folder");
    await folderItem.SaveAsync();

    var calendarItem = folderItem.NewCalendar("NewCalendar");
    calendarItem.Dates.Add(new StiCalendarDate("Monday", StiDaysOfWeek.Monday));
    await calendarItem.SaveAsync();
}
```

Getting Item

To get an existing item is necessary to know the key that is passed to the method as a parameter. Use methods `StiItem.Get()` and `StiItem.GetAsync()`:

```
public void GetCalendarItem()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var item = connection.Items.Get("CalendarItemKey");
    if (item != null)
    {
        var calendarItem = item as StiCalendarItem;
        if (calendarItem != null)
        {
            var calendarItemDescription = calendarItem.Description;
        }
    }
}
```

Asynchronous method:

```
public async void GetCalendarItemAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var item = await connection.Items.GetAsync("CalendarItemKey");
    if (item != null)
    {
        var calendarItem = item as StiCalendarItem;
        if (calendarItem != null)
        {
```

```
            var calendarItemDescription = calendarItem.Description;
        }
    }
}
```

Note that the result must cast to the required type and check for null.

Getting a List of Items

To find or process items, there is a method that allows you to get a list of all objects `StiItem`, to which the current user can access. Use the method `FetchAll()` (`FetchAllAsync()`).

```
public void ProcessItems()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");
    var items = connection.Items.Root.FetchChilds();

    //find folder with name "Folder1"
    var folder1 = items.First(a => a.Name == "Folder1");
}
```

Asynchronous method:

```
public async void ProcessItemsAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");
    var items = await connection.Items.Root.FetchChildsAsync();

    //is exist any folder
    var isFolder = items.Any(a => a.IsFolder);
}
```

Deleting Item

Remove element by calling `Delete()` (`DeleteAsync()`):

```
public void DeleteItem()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var item = connection.Items.Get("CalendarItemKey");
    if (item != null)
    {
        //Delete item with skipping undeletable items
        item.Delete(true, true);
    }
}
```

Asynchronous method:

```
public async void DeleteItemAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var item = await connection.Items.GetAsync("CalendarItemKey");
    if (item != null)
    {
        //Delete item without moving it into the recycle bin
        await item.DeleteAsync(false);
    }
}
```

Resource Item

Some element types (`StiFileItem, StiReportSnapshotItem, StiReportTemplateItem`) include resources. Using the methods `UploadFromFile(), UploadFromFileAsync(),UploadFromArray(), UploadFromArrayAsync(), DownloadToFile(), DownloadToFileAsync(), DownloadToArray(), DownloadToArrayAsync()` You can manipulate the data contained in these resources.

For example, creating a file on the server with the loading data into it looks like this:

```
public void CreateNewFile()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var content = File.ReadAllBytes(@"C:\testfile.xml");

    var newFile = connection.Items.Root.NewFile("TestFile.xml");
    newFile.Save();

    newFile.UploadFromArray(content);
}
```

Asynchronous method:

```
public async void CreateNewTemplateAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var newTemplate = connection.Items.Root.NewReportTemplate("Master-Detail");
    await newTemplate.SaveAsync();

    await newTemplate.UploadFromFileAsync(@"C:\master-detail.mrt");
```

```
}
```

Loading item from the server and saving it to a file on the local computer as follows:

```
public void DownloadFile()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var newFile = connection.Items.Get("FileItemKey");
    if (newFile is StiFileItem)
    {
        var data = (newFile as StiFileItem).DownloadToArray();
        File.WriteAllBytes(@"c:\newfile", data);
    }
}
```

Asynchronous method:

```
public async void DownloadFileAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var report = await connection.Items.GetAsync("ReportTemplateItemKey");
    if (report is StiReportTemplateItem)
    {
        await (report as StiReportTemplateItem).DownloadToFileAsync(@"C:\Master-
Detail.mrt");
```

Running Report

The result of creating Report Template in the tree is a new element type `StiReportTemplateItem`. This object describes the report without data. Designing items of the report is made through the Navigator interface, but managing the construction is possible by means of appropriate methods `StiReportTemplateItem.Run()` and `StiReportTemplateItem.RunAsync()`.

The parameter specifies the item `StiReportTemplateItem` or `StiFileItem` where you saved the rendered report. The object must be created before saving.

The item `StiFileItem` can be one of the possible types of enumeration `StiFileType` (ReportSnapshot, PDF, XPS, PowerPoint, HTML, Text, RichText, Word, OpenDocumentWriter, Excel, OpenDocumentCalc, Data, Image, XML, XSD, CSV, DBF). One of these values are specified in the method `StiServerConnection.Items.Root.NewFile()` as a parameter.

The following example creates a report template, loads data into it and runs the report, after which the result is stored in the report snapshot:

```
public void RunReportToShapshot()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var reportTemplateItem = connection.Items.Root.NewReportTemplate("report-
template");
    reportTemplateItem.Save();
    reportTemplateItem.UploadFromFile(@"C:\report.mrt");

    var reportSnapshotItem = connection.Items.Root.NewReportSnapshot("report-
snapshot");
    reportSnapshotItem.Save();
    reportTemplateItem.Run(reportSnapshotItem);
}
```

Asynchronous method creates a report template, converts data and runs the report, after which the result is stored in the PDF file:

```
public async void RunReportToPdfFileAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var reportTemplateItem = connection.Items.Root.NewReportTemplate("report-
template");
    await reportTemplateItem.SaveAsync();
    await reportTemplateItem.UploadFromFileAsync(@"C:\report.mrt");

    var pdfReportItem = connection.Items.Root.NewFile("report.pdf", StiFileType.Pdf);
    await pdfReportItem.SaveAsync();
    await reportTemplateItem.RunAsync(pdfReportItem);
}
```

Exporting Report Snapshot

The result of the construction of the report from `StiReportTemplateItem` is `StiReportSnapshotItem`. Export data from it can be done by methods `StiReportSnapshotItem.Export()` and `StiReportSnapshotItem.ExportAsync()`.

The data is stored in the item `StiFileItem`. It should be created in advance with a specific type the constructor, as described in the previous chapter.

Optional parameter method `StiReportSnapshotItem.Export()` is an object `StiTextExportSet`, which describes many options of visualization in the report to

export.

This example demonstrates exporting the report snapshot to the Excel Document:

```
public void ExportSnapshotToExcel()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var reportSnapshotItem = connection.Items.Root.NewReportSnapshot("report-
snapshot");
    reportSnapshotItem.Save();
    reportSnapshotItem.UploadFromFile(@"C:\report-snapshot.mdc");

    var fileItemExcel = connection.Items.Root.NewFile("ExcelDocument",
StiFileType.Excel);
    fileItemExcel.Save();

    reportSnapshotItem.Export(fileItemExcel,
        new StiTextExportSet
        {
            BorderType = Report.Export.StiTxtBorderType.UnicodeDouble,
            CutLongLines = true,
            KillSpaceGraphLines = true
        });
}
```

An example of an asynchronous method. Exporting a report snapshot to the XML file:

```
public async void ExportSnapshotToXmlAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var reportSnapshotItem = connection.Items.Root.NewReportSnapshot("report-
snapshot");
    await reportSnapshotItem.SaveAsync();
    await reportSnapshotItem.UploadFromFileAsync(@"C:\report-snapshot.mdc");

    var fileItemXml = connection.Items.Root.NewFile("XmlFile", StiFileType.Xml);
    await fileItemXml.SaveAsync();

    await reportSnapshotItem.ExportAsync(fileItemXml);
}
```

Scheduling actions

Stimulsoft Reports Server supports flexible system schedulers that allow the automation of various actions and events schedule. The object `StiSchedulerItem` inherited from `StiItem` is used to work with the scheduler.

After creating an instance of this object, you must add actions using the method `StiSchedulerItem.AddRunReportAction()`.

The parameters of this overloaded method take several sets of objects that define the functionality of the scheduler. The minimum set of parameters is as follows:

`AddRunReportAction(StiReportTemplateItem reportTemplateItem, StiReportSnapshotItem reportSnapshotItem)` – at work scheduler built report template `reportTemplateItem` and stored in a snapshot `reportSnapshotItem`;

`AddRunReportAction(StiReportTemplateItem reportTemplateItem, StiFileItem fileItem)` – at work scheduler built report template `reportTemplateItem` and exported to a file `fileItem`.

To start the scheduler uses the `StiSchedulerItem.Run()` (`StiSchedulerItem.RunAsync()`) method.

This example creates a scheduler to render the report in a snapshot:

```
public void CreateSchedulerRunReportToSnapshot()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    // Create folder
    var folderItem = connection.Items.Root.NewFolder("folder");
    folderItem.Save();

    // Create report template
    var reportTemplateItem = folderItem.NewReportTemplate("report-template");
    reportTemplateItem.Save();
    reportTemplateItem.UploadFromFile(@"c:\ReportTemplate.mrt");

    // Create report snapshot
    var reportSnapshotItem = folderItem.NewReportSnapshot("report-snapshot");
    reportSnapshotItem.Save();

    // Create scheduler
    var schedulerItem = folderItem.NewScheduler("scheduler", StiSchedulerIdent.Once);
    schedulerItem.AddRunReportAction(reportTemplateItem, reportSnapshotItem);
    schedulerItem.Save();

    // Run scheduler
    schedulerItem.Run();

    connection.Users.Logout();
}
```

This asynchronous method creates a scheduler which renders the template to the PDF file:

```
public async void CreateSchedulerRunReportToSnapshotAsync()
```

```
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    // Create folder
    var folderItem = connection.Items.Root.NewFolder("folder");
    await folderItem.SaveAsync();

    // Create report template
    var reportTemplateItem = folderItem.NewReportTemplate("report-template");
    await reportTemplateItem.SaveAsync();
    await reportTemplateItem.UploadFromFileAsync(@"c:\ReportTemplate.mrt");

    // Create file
    var fileItem = folderItem.NewFile("file", StiFileType.Pdf);
    await fileItem.SaveAsync();

    // Create scheduler
    var schedulerItem = folderItem.NewScheduler("scheduler", StiSchedulerIdent.Once);
    schedulerItem.AddRunReportAction(reportTemplateItem, fileItem);
    await schedulerItem.SaveAsync();

    // Run scheduler
    await schedulerItem.RunAsync();

    connection.Users.Logout();
}
```

1.3 Roles

Roles

Stimulsoft Reports Server supports a Role-based Access Control. Role is an object that determines the level of user access to system resources. Members having the same role have the same rights of access to system objects.
The most important property of the role is `StiRole.Permissions`. It is an object of the class `StiRolePermissions` that describes a set of permissions (`StiPermissions`) for system objects.
`StiPermissions` enumeration includes the following options:
`None` (Denies all);
`Create` (Allows creating an item);
`Delete` (Allows deleting an item);
`Modify` (Allows modifying an item);
`Run` (Allows running an item);
`View` (Allows viewing an item);

`ModifyView` (Allows modifying and viewing an item);
`CreateDeleteModifyView` (Allows creating, deleting, modifying and viewing an item);
`RunView` (Allows running and viewing an item);
`All` (Allows any action with an item).

In order to use the class `StiUserConnection` you should create an instance of `StiServerConnection` and call one of its methods (`StiServerConnection.Roles`).

Creating and Saving Roles

To create a new role, you need to log in with a user name that has permission to work with roles, and create an object of the type `StiRole`, and then call its method `StiRole.Save()` (`StiRole.SaveAsync()`):

```
public void CreateNewRole()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var role = connection.Roles.New("UserRole");
    role.Permissions = connection.Roles.ManagerRole.Permissions;
    role.Permissions.SystemMonitoring = StiPermissions.RunView;
    role.Save();

    connection.Users.Logout();
}
```

Asynchronous example:

```
public async void CreateNewRoleAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var role = connection.Roles.New("CustomRole");
    role.Permissions.SystemUpdate = StiPermissions.All;
    role.Permissions.ItemSchedulers = StiPermissions.CreateDeleteModifyView;
    await role.SaveAsync();

    await connection.Users.LogoutAsync();
}
```

Getting Role Info

In order to obtain information about the role, use the methods `GetByKey()` or `GetByKeyAsync()` that return an object `StiRole`.
However, a more convenient way, without requiring the key is the use of one of the objects

`StiServerConnection.Roles.AdministratorRole`,
`StiServerConnection.Roles.ManagerRole`,
`StiServerConnection.Roles.UserRole`.
Before calling this methods you must log in as a user whose rights are allowed to have access to the necessary information.

```
public void GetRoleInfo()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var userRole = connection.Roles.GetByKey("RoleKey");
    var roleDescription = userRole.Description;

    connection.Users.Logout();
}
```

Asynchronous method:

```
public async void GetRoleInfoAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var managerRole = connection.Roles.ManagerRole;
    var managerCreatedDate = managerRole.Created;
    var managerPermissions = managerRole.Permissions;

    await connection.Users.LogoutAsync();
}
```

Getting a List of Roles

To find or process information about roles of the system, there is a method that allows you to get a list of all objects `StiRole`, to which the current user can access. Use the method `FetchAll()` (`FetchAllAsync()`).

```
public void ProcessUsersInfo()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    var roles = connection.Roles.FetchAll();

    //find role with full access to system monitoring
    var monitoringAdministrator = roles.First(a => a.Permissions.SystemMonitoring ==
StiPermissions.All);

    connection.Users.Logout();
}
```

Asynchronous example:

```
public async void ProcessUsersInfoAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    var roles = await connection.Roles.FetchAllAsync();

    //is exist role with name "ReportAdministrator"
    var isReportAdministrator = roles.Any(a => a.Name == "ReportAdministrator");

    await connection.Users.LogoutAsync();
}
```

Deleting Role

Remove the role by calling `Delete()` (`DeleteAsync()`):

```
public void DeleteRole()
{
    var connection = new StiServerConnection("localhost:40010");
    connection.Users.Login("UserName@example.com", "Password");

    connection.Roles.Delete("RoleKey");

    connection.Users.Logout();
}
```

Asynchronous method:

```
public async void DeleteRoleAsync()
{
    var connection = new StiServerConnection("localhost:40010");
    await connection.Users.LoginAsync("UserName@example.com", "Password");

    await connection.Roles.DeleteAsync("RoleKey");

    await connection.Users.LogoutAsync();
}
```