PNMsoft Knowledge Base

Sequence Developer Guides

Authentication

Dec. 2013
Product Version 7.0 and above

PNMSOFT
Business Processes in Motion™

TABLE OF CONTENTS

General Document Information

Purpose

The purpose of this document is to enable Developers to establish authentication and impersonation with Sequence via external systems, in order to invoke Sequence API functions.

Prerequisites

- Experience with .NET.
- Familiarity with Sequence API.
- Familiarity with the Sequence configuration files.

Overview

Sequence can be operated via its native Administration and Flowtime (Runtime) environments. Alternatively, Sequence can be operated via its API. In order to invoke Sequence API via external systems, you must first perform authentication with Sequence.

Steps Required Before Invoking the API

In order to invoke the API, the external system must first perform the following steps:

1. **Authentication:** authenticate itself with Sequence.
2. **Impersonation:** establish impersonation using the credentials provided in Authentication, in order to run API code.
3. **Invoke** API methods.

.This guide will describe how to perform these steps.

Authentication Providers

Sequence performs Authentication according to a chain of registered authentication providers. These are defined in the configuration file in the following section:

```
<sequence.engine>
     <authentication>
         <providers>
```

For example:

```
      <providers>
        <add
type="PNMsoft.Sequence.Security.WindowsAuthenticationProvider,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1b1b90c1d5dca1">
        </add>
        <add
type="PNMsoft.Sequence.Security.UsernameAuthenticationProvider,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1b1b90c1d5dca1">
        </add>
      </providers>
```

Sequence attempts to authenticate the first provider. If the authentication does not succeed, then it moves on to the next one.

For example:

1. Windows authentication attempted and failed.
2. Username/Password authentication attempted and succeeded.

Authentication Types

Sequence supports the following authentication types:

- Windows Authentication - `PNMsoft.Sequence.Security.WindowsAuthenticationProvider`
- Username/Password - `PNMsoft.Sequence.Security.UsernameAuthenticationProvider`
- Claims Identity - `PNMsoft.Sequence.Security.ClaimsIdentityAuthenticationProvider`
- Membership (user management based on Microsoft membership provider) - `PNMsoft.Sequence.Security.MembershipAuthenticationProvider`
- Anonymous - `PNMsoft.Sequence.Security.AnonymousUserAuthenticationProvider`

You can also write a Custom Authentication provider. You would need to add this Custom provider to the configuration file.

Some authentication types may require additional configuration.

Implementing Authentication in a Web Application

Sequence includes an out-of-the-box HttpModule for Authentication and Impersonation, called: **SequenceAuthenticationModule**.

SequenceAuthenticationModule takes care of all aspects of Authentication and Impersonation for you.

SequenceAuthenticationModule is contained in the Sequence web.config file.

Depending on the mode of your IIS application pool, SequenceAuthenticationModule is defined in a different section of web.config, as follows:

- If your IIS application uses classic application pool mode, SequenceAuthenticationModule is in:

  ```
  <system.web>
      <httpModules>
  ```

- If your IIS application uses integrated application pool mode, SequenceAuthenticationModule is in:

  ```
  <system.webServer>
      <Modules>
  ```

For example:

```
    <httpModules>
      <add name="SequenceAuthenticationModule"
type="PNMsoft.Sequence.Web.AuthenticationModule,
PNMsoft.Sequence.Web, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
    </httpModules>
```

To get the current user (if you are using the SequenceAuthenticationModule):

1. In your code, add one of the following methods:

   ```
   WorkflowRuntime.Engine.GetCurrentUserWithCheck()
   SecurityManager.CurrentAuthenticated
   ```

Note: It is recommended not to manage the authentication independently, rather to use the SequenceAuthenticationModule module. Only in specific cases where the module is inappropriate, you can write custom code which manages authentication and impersonation.

Authenticating Independently in your Code

While it is recommended to use SequenceAuthenticationModule for authentication, your solution may require custom authentication (if SequenceAuthenticationModule does not cover your scenario).

This section describes how to authenticate without using SequenceAuthenticationModule.

As noted in the Overview, you must perform both Authentication and Impersonation before you can invoke the API.

Performing Authentication

To authenticate independently, you need to use the IAuthenticationService.

For example:

```
AuthenticatedUser user =
WorkflowRuntime.Engine.GetService<IAuthenticationService>().Authenticate();
```

Using this service, you can authenticate using one of the following options.

```
Authenticate()
```

```
Authenticate (string username, string password)
```

```
Authenticate(AuthenticationToken token)
```

Select one of the above options based on the Authentication type you are using, according to the following guidelines:

1. Use `Authenticate()` for Authentication types which do not require additional authentication parameters, for example:

   - Windows
   - Membership Provider
   - Claims

   In these cases, Sequence can take the parameters from the context in which the application runs, and the provider can create the token independently.
2. Use `Authenticate(string username, string password)` for Username/Password authentication, where you need to provide the username and password as parameters.
3. Use `Authenticate(AuthenticationToken token)` for a custom authentication type that requires a token (e.g. worker ID number) which must be passed as a parameter.

Note: `AuthenticationToken` is an object that contains the authentication information that is used by the application pool.

As a result of the Authentication, an object of type `AuthenticatedUser` is returned. This object contains data about the Sequence user.

Note: this user must already exist in the Sequence database.

Performing Impersonation

You use the SecurityManager object from PNMsoft.Sequence.Security namespace to perform impersonation. The object that is returned is of type `SecurityContext`, that encapsulates security-related data.

To perform independent impersonation:

1. Create a `SecurityContext` object.
2. Run API function(s).
3. Destroy the `SecurityContext` using the `Dispose()` function. (This is an important security measure, otherwise you leave impersonation open).

   Note: You can use the `using` statement to Open and Destroy the `SecurityContext`.

See *An Example* below.

A Note About the .NET IPrincipal Object

In .NET, all code runs on a specific thread. It is possible to attach a `Principal` object to the thread. Sequence code can run with or without attaching the `SecurityContext` object to the thread.

You can use one of two methods defined on the `SecurityManager` object:

- `CreateContext`: Does not attach the `IPrincipal` object to the current thread.
- `Impersonate`: the `IPrincipal` object is attached to the current thread.

  Note: the `AuthenticatedUser` implements IPrincipal interface.

For Example:

- Use this code when you want to attach user object to the current thread

```
AuthenticatedUser user =
WorkflowRuntime.Engine.GetService<IAuthenticationService>().Authenticate();
            using (SecurityContext securityContext =
SecurityManager.Impersonate(user))
{
                //add your API calls here
}
```

Else

- Use this code:

```
AuthenticatedUser user =
WorkflowRuntime.Engine.GetService<IAuthenticationService>().Authenticate();
            using (SecurityContext securityContext =
SecurityManager.CreateContext(user))
{
                //add your API calls her
}
```

An Example

You writing a webpage and you wish to invoke several Sequence API methods when the page loads. In this case, you can create the security context at the beginning and destroy it at the end using the `Dispose()` method, and you can place several API calls in the middle.

```
using System;

using System.Web.UI;

using PNMsoft.Sequence.Runtime;

using PNMsoft.Sequence.Security;


namespace SequenceEx.Security

{

    public class MyPage : Page

    {

        private SecurityContext securityContext;


        protected override void OnInit(EventArgs e)

        {

            AuthenticatedUser user =
WorkflowRuntime.Engine.GetService<IAuthenticationService>().Authenticate();

            this.securityContext = SecurityManager.Impersonate(user);


            base.OnInit(e);

        }


        protected override void OnPreRender(EventArgs e)

        {

            base.OnPreRender(e);

        }


        protected override void Render(HtmlTextWriter writer)

        {

            base.Render(writer);

        }


        public override void Dispose()

        {

            if (this.securityContext != null)

            {

                this.securityContext.Dispose();

                this.securityContext = null;
            }

            base.Dispose();
        }
    }
}
```