Documentation of the Assistant module "External Communication"

FKE

November 6, 2018

Contents

1 Description

The service ExternalCommunication serves as a component for sending and receiving SignalR messages. The XClient which renders the user interface, connects to the endpoint provided by External Communication via SignalR. By using the provided endpoints a user can control the KPUs in a generic and modular way, depending on the interfaces, models and views provided by the KPUs themselves. For this, External Communication communicates with the following services:

- Core Service
  - active: interaction with the KPUs which are logically 'beyond' the Core Service
  - passive: returning requested KPU-packages that are necessary in the client to 'display a KPU'
- Security Service
  - active: user authentication and immanent permission checks on every query from a client
- Presenter Service
  - active: registration of a connection for the model updates of a model
  - passive: forwarding of packaged model updates to clients

2 Usage

External Communication is the component used for communication with SignalR clients. SignalR Core is used which is compliant to netstandard so it can be used both in .Net Framework and .Net Core applications. The interfaces client -> server and server -> client used for External Communication are defined in the project "AssistantViewerInterfaces" of the Assistant solution.

- - IAssistantViewer: this is the interface for methods the client has to offer to the server on its side. For the server to actually be able to call them, a little more boilerplate code is necessary, more on that later.
- - IExternalCommunication: this is the interface for methods, the server offers to the client. Notable examples include:
  - Login: user authentication and association with a certain connection

- RequestExecute: Execution of an action on a KPU
- ReceiveViewRequest: Request for a package that a client needs to have and use in order to be able to display a KPU
- ReceiveSubscribeRequest / ReceiveUnsubscribeRequest: Request for the subscription to Model Updates from a certain KPU

To receive "messages" from the server, it's necessary to both implement the corresponding methods as well as let the SignalR-client object know about them. Listing 1 shows the class SignalRClient as an example implementation for the hub configuration and registration.

```
// missing code
using Microsoft.AspNetCore.SignalR.Client;
// missing code
class SignalRClient
{
private HubConnection Hub;

public async Task CreateConnection()
{
var connectionString = "http://127.0.0.1:4242/ecom";
Hub = new HubConnectionBuilder()
.WithUrl(connectionString)
.Build();
RegisterMethods();
await Hub?.StartAsync();
}
private void RegisterMethods()
{
Hub?.On(nameof(OnReceiveBadLogin), async () => await
    OnReceiveBadLogin());
Hub?.On(nameof(OnReceiveGoodLogin), async () => await
    OnReceiveGoodLogin());
Hub?.On(nameof(OnBatchDataReceived), async (string viewId,
    string[] items, string[] data, DateTime[] dateTimes) => await
     OnBatchDataReceived(viewId, items, data, dateTimes));
Hub?.On(nameof(OnMessageReceived), async (string message, string
    [] buttons) => await OnMessageReceived(message, buttons));
Hub?.On(nameof(OnPackageReceived), async (string viewId, string
    data) => await OnPackageReceived(viewId, data));
Hub?.On(nameof(OnDisconnect), async () => await OnDisconnect());
}
// missing code
}
```

Listing 1: SignalR Hub creation

In this example, two methods are shown, "Connect" and "RegisterMethods". Connect attempts establish the connection with the SignalR server and stores the connection object in the property "Hub". This object can then be used to communicate with the server. "RegisterMethods" informs the connection object about how to handle certain calls that are incoming

from the server. Calls in SignalR in both directions are essentially equivalent to calling methods in a program which can have parameters. However, the data is only transferred serialized via SignalR, meaning the server wants to call on the client results in little more than the name of that method being transferred as a string to the client. (it's a bit more complicated but this is effective in wrapping one's head around the concept). To know during runtime, what to do with a call from the server that starts with "OnReceive-GoodLogin", that information must be known to the Hub as can be seen in Listing 2

```
1 Hub?.On(nameof(OnReceiveGoodLogin), async ()=> await
      OnReceiveGoodLogin());
```

Listing 2: SignalR server callback method registration

The method with the same name is being called via a lambda expression to represent the behavior for the incoming call. How exactly the method is implemented is dependent on how the client is built, for example one could switch the view from the login screen to some menu display for all available KPUs. Another one of those details of implementation is that the method implemented in the client is named equal to the SignalR call which enables the use of the "nameof"-Operator which replaces the method name with its string representation during compile time. The opposite way of sending messages, from client to server is also done through the Hub, as shown in Listing 3

```
1 public async Task LoginAsync(string user, string password)
2 {
3     await Hub.InvokeAsync("Login", user, password);
4 }
```

Listing 3: SignalR client-side server method invocation

Here, the parameters for user name and password are simply passed on to the call to the server side method called "Login".

Note: For methods with return actual values, Hub.On and Hub.InvokeAsync are written generically so almost any kind of method / lambda function can be used.

3 Installation

External Communication is installed by deploying it within the Service Fabric Cluster Application "Assistant".

4 Configuration

To configure the service's logging, one can alter the Nlog.xml to switch to a different target / layout.[1]

5 Dependencies

All other services of the Service Fabric Application "Assistant" also have to be deployed onto the cluster. The Service Fabric Application "Access Control" must also be deployed and accessible on the cluster. External Communication requires the following libraries which are not available from the standard nuget server:

- BreanosConnectors.Kpu.Communication.Common

6 Requirements

For External Communication to be used, port 4042 of the Node's system it's deployed on must be accessible. At present, External Communication must be installed on exactly one Node of the cluster. This is because External Communication is designed as a "stateless service" with the Service Fabric Application and the forwarding logic for model updates wouldn't work if it were running on more than one node. Logging is implemented via NLog which can utilize an existing database as the logging target which then needs an accessible DBMS with writable database.

[1] https://github.com/nlog/NLog/wiki/Configuration-file

Listings