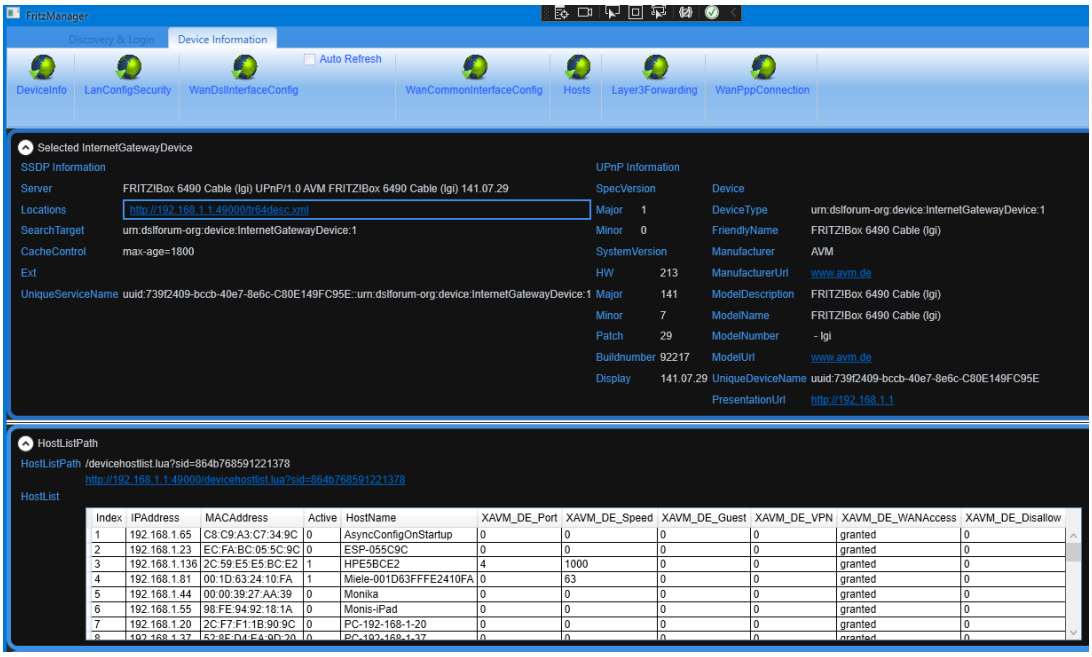
Aufbau des Programms RS.Fritz:Manager Februar/März 2022

[Rans4ckeR/RS.Fritz.Manager: A Windows .NET app with WPF UI to read information from and manage FritzBox devices using pure WCF calls. (github.com)]



Start des Programms

Es wird zuerst in der Methode **App** von **App.xaml.cs** die methode **Host.CreateDefaultBuilder ()** aufgerufen um den Host zu initialisieren. Hierbei werden die ViewModels, die Services und der **FritzServiceOperationHandler** bekanntgemacht.

In App.xaml.cs wird dann die methode **OnStartUp** aufgerufen.

In dieser Methode werden dann mit:
`var mainWindow = host.Services.GetRequiredService<MainWindow>();`
die Konstruktoren der erforderlichen services aufgerufen.

Zuerst wird der **FritzServiceOperationHandle**r (Ordner .Domain/SoapClient) instantiiert.

Dann wird die Klasse DeviceLogInInfo (Ordner .UI/Entities) instantiiert und diesem der **FritzServiceOperationHandle**r als Parameter mitgegeben.

FritzServiceViewModel (Ordner .UI/Infrastructure) wird instantiiert, dieses ViewModel wird dann z.B. allen UserControls als **base** Klasse mitgegeben. **FritzServiceViewModel werden** ‚DeviceLoginInfo' und ‚FritzServiceOperationHandler' als Parameter mitgegeben.

Nun geht's weiter in App.xaml.cs und das GUI wird aufgebaut.

Erster User Befehl(Discover Internet Gateway Devices)

Das Anklicken des Buttons ‚Discover Internet Gateway Devices' → führt zur Ausführung von DoExecuteDefaultCommandAsync() im MainWindowViewModel. Es wird der deviceSearchService aufgerufen, der die die gefundenen Gateway Devices in die Observable Collection ‚ObservableInternetGatewayDevice' einträgt.
Außerdem wird aus der DeviceSearchService (Ordner .Domain/Discovery) Klasse in der Task ‚GetDevicesAsync' die Task ‚GetUPnPDescription' aufgerufen. Hier wird die Datei ‚tr64desc.xml' geladen.

Zweiter User Befehl (Select A Discovered Device)

Die Auswahl des Internet Gateway Devices in 'Select A Discovered Device' bewirkt in ‚DeviceLogInfo.cs' (Ordner .Ui/Entities) in der Methode Receive(PropertyChangedMessage<ObservableInternetGatewayDevice?> message)

- Eintragen des ausgewählten InternetGatewayDevice in den fritzServiceOperationHandler
- Abrufen des Security ports für https Transport von der Fritzbox
- Abrufen der User Liste von der Fritzbox

Nebenbemerkung: Die Umschaltung der UserControls erfolgt wohl in MainWindowViewModel

```
public ObservableObject? ActiveView { get => activeView; set => _ =
SetProperty(ref activeView, value); }
```

In der Info Anzeige kann man nun auf den Link klicken. Der angezeigte Path öffnet sich im Browser. Hierzu dient die Klasse **ExternalBroserHyperlink** (im Ordner .UI/Infrastructure/

Weitere User Controls

Im User Interface sind Buttons für die verschiedenen Services angelegt.

Beim Clicken eines Buttons wird in der **…ViewModel.cs** Datei des zugeordneten **UserControls** die Task DoExecuteDefaultCommandAsync() gestartet und darin werden die gewünschten Aktionen des entsprechenden Services gestartet.

Einer **…Response** Variablen wird das Ergebnis der Aktion zugewiesen.

Die Aktionen befinden sich in der Klasse **FritzServiceOperationHandler** im Ordner ‚**SoapClien**t'.

FritzServiceOperationHandler erbt von ServiceOperationHandler und IFritzServiceOperationHandler.

Der Aufruf der Aktion wird dann an ExecuteAsync() übergeben. Vorher wird durch Aufruf von ‚GetFritzHostserviceClient() ein FritzHostServiceClient erstellt. Für die Erzeugung des FritzHostServiceClient werden als Parameter die url http://192.168.1.1:49000/tr64desc.xml in der sich die Beschreibung des Service

befindet, ein Flag (http oder https), die Control URL (z.B. upnp/Control/hosts), der security port (49443) und die NetworkCredentials mitgegeben.

Es kommt dann ein voll ausgestatteter Client mit z.B. in der folgenden Form zurück: https://192.168.1.1:49443/upnp/control/hosts .

Dieser Client führt dann die gewünschte Aktion aus und liefert die Response zu der gewünschten Aktion zurück.

Die Response befindet im ViewModel, die mit dem View durch Binding verknüpft ist.

So erscheint die Rückmeldung in der Benutzeroberfläche.

Darstellung der angezeigten Daten:

In App.Xaml Befinden sich DataTemplates, hier muss ein neues Datatemplate hinzugefügt werden, z.B.:

```
<DataTemplate DataType="{x:Type
domain:WanCommonInterfaceConfigGetTotalBytesReceivedResponse}">
                <Border Background="{StaticResource ForegroundSolidColorBrush}"
BorderBrush="{StaticResource TitleBackgroundSolidColorBrush}" BorderThickness="5"
CornerRadius="10" Padding="5">
                    <Expander Foreground="{StaticResource TextSolidColorBrush}"
IsExpanded="True" Header="TotalBytesReceived">
                        <Grid>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="Auto"/>
                            </Grid.RowDefinitions>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="Auto"/>
                                <ColumnDefinition Width="Auto"/>
                            </Grid.ColumnDefinitions>
                            <Label Grid.Row="0" Grid.Column="0" Target="{Binding
ElementName=TotalBytesReceived}" Content="_TotalBytesReceived"/>
                            <TextBlock x:Name="TotalBytesReceived" Grid.Row="0"
Grid.Column="1" Text="{Binding TotalBytesReceived}"/>
                        </Grid>
                    </Expander>
                </Border>
            </DataTemplate>
```

Erstellung eines neuen Service:

- In UserControls neuen Ordner erstellen mit Namen des Service
- Von bereits vorhandenem UserControl mit ähnlichem Aufbau die …View.xaml Datei, die …View.xaml.cs und …ViewModel.cs Datei rüberkopiern, die Dateien entsprechend dem neuen Service umbenennen und den Inhalt anpassen.
- In der Datei MainWindowsViewModel.cs das neue ViewModel im Constructor als Parameter aufnehmen, public Variablen für das neue ViewModel anlegen und im Constructor die public Variablen initialisieren.
- Die Task DoExecuteDefaultCommandAsync() solange noch keine Aktionen definiert sind mit dummy Befehl z.B. ‚await Task.Delay(1000);‘ besetzen

und die weiteren Funktionen zunächst auskommentieren. Danach sollte sich das Programm schon kompilieren lassen.

- In RS.FritzManager.Domain/FritzServices/Services neuen Ordner mit dem Namen des Service anlegen. Dann den gesamten Inhalt des entsprechenden Ordners eines anderen ähnlichen Service hineinkopieren und den Inhalt anpassen bzw. die Dateinamen ändern und die nicht benötigten Entities löschen.

- Im Ordner **SoapClient**
    1. Private Variable für den Service erstellen und im Constructor als Parameter anlegen und zuordnen. Dann weiter wie unten für das Anlegen einer neuen Aktion beschrieben.
- In App.xaml
    1. DataTemplate hinzufügen, z.B.

```
<DataTemplate DataType="{x:Type ui:HostsViewModel}">
   <ui:HostsView/>
</DataTemplate>
```

- In App.xaml.c unter host = Host.CreateDefaultBuilder()
das neue ViewModel und die ClientFactory eintragen

- In MainWindow.xaml in RibbonTab (Header="Device Information") das neue Button Element einfügen.

<u>Hinzufügen einer neuen Aktion zu einem bereits vorhandenen Service:</u>

- In RS.FritzManager.Domain/FritzServices/Services/*My..ServiceName*/Entities
    1. Neue Response Datei anlegen.
    2. Neue Request Datei anlegen.
    3. Eintrag in ‚IFritz..*My..ServiceName..*Service.cs'
    4. Eintrag in ‚Fritz..*My..ServiceName..*Service.cs'

- Im Ordner **UserControls** in ***My..Servicename..*** **ViewModel .cs:**
    1. in der Task DoExecuteDefaultCommandAsync() den neuen Befehl eintragen, z.B.: GetWanCommonInterfaceConfigGetCommonLinkPropertiesAsync()
    2. Die zugehörige Task anlegen.
    3. Die private Variable anlegen.
    4. Die public Variable anlegen (für MVVM Aktionen)

- Im Ordner **UserControls** in ***My..Servicename..*** **View.xaml:**

    1. **ContentControl** für neue **Grid.Row** mit Binding eintragen

- Im Ordner **SoapClient**
    1. In 'IFritzServiceOperationHandler.cs' Task eintragen

2. In ‘FritzServiceOperationHandler.cs’ Task eintragen, private Variable eintragen und für jeden neuen Service ein Interface definieren

- In der Datei **App.xaml**
    1. Ein neues DataTemplate anlegen (anderes als Vorlage einkopieren) und analog zur Vorlage ausfüllen
-

Und die zugehörige Task anlegen:

```
private async Task GetWanCommonInterfaceConfigGetCommonLinkPropertiesAsync()
{
    wanCommonInterfaceConfigGetCommonLinkPropertiesResponse = await
    FritzServiceOperationHandler.GetWanCommonInterfaceConfigGetCommonLinkPropertiesAsync();
}
```

Es muss auch die private Variable vom Typ WanCommonInterfaceConfigGetCommonLinkPropertiesResponse? angelegt werden.

Die Definition des Typs erfolgt in RS.FritzManager.Domain/FritzSservices/…z.B. WanCommonInterfaceConfig/Entities. Hier wird eine eine neue Datei angelegt. Z.B: WanCommonInterfaceConfigGetCommonLinkPropertiesResponse.cs.

Darin die Definition:

```
public sealed record WanCommonInterfaceConfigGetCommonLinkPropertiesResponse
```

und die Anlage für alle Parameter, die aus der Soap Response ausgelesen werden sollen.

Nun kann man schon einmal überprüfen, ob alles kompiliert und läuft, indem man in der ViewModel Datei statt des normalen Befehls *await Task.Delay(1000);* vorübergehend einträgt.

- Eintragen des neuen Befehls in: IFritzServiceOperationHandler und FritzServiceOperationHandler

- In RS.Fritz.Manager.Domain\FritzServices\Services\WanCommonInterfaceConfig\Entities\ neue Requestdatei z.B. WanCommonInterfaceConfigGetCommonLinkPropertiesResponse.cs

- Eintragen in z.B.:
- IFritzWanCommonInterfaceConfigService und FritzWanCommonInterfaceConfigService