VisiWin7
HMI-SOFTWARE

Introduction to VisiWin 7

INOSOFT
HMI UNLIMITED

Contents

1 About this documentation

This documentation [1] [2] provides information on the prerequisites for the operation of VisiWin 7 (licensing and system requirements), instructions on how to install VisiWin 7 and a step-by-step tutorial for new users.

Special information that needs your attention is highlighted using the following symbols:

Attention:
Important notice to avoid exceptions

Important:
Important notice to avoid inconveniences

Note:
Additional information

Tip:
Useful function or option for optimization

Example:
An example illustrating the described concept, function, or feature

1. This documentation is regularly proofread to ensure that the contents agree with the described software. However, deviations cannot be ruled out in the light of agile software development. Therefore, we cannot guarantee the correctness and completeness of this documentation.
2. Last update: December, 2021

2 Licensing

License types

VisiWin 7 is available under two basic types of license, which are subject to fees:

IDE

Projects can be created and edited in the IDE. The IDE can be installed on any number of computers. However, a *developer license* must be purchased for each developer. If the USB dongle is used as a license mode, the dongle may be passed on to another user for the duration of the license.

Runtime

When finished, a project is ready to be installed on a target computer for use by a machine operator. The target computer must have a *runtime license*. The runtime license depends on the functional scope of the project.

License mode

There are two license modes available to unlock a license:

USB Dongle

A USB dongle is a piece of hardware the size of a memory stick you can plug into a USB port. A USB dongle contains the license data. A dongle driver is installed during the installation of the software. However, you may have to update the dongle driver manually due to Windows updates. If the IDE starts in demo mode although you have plugged in the USB dongle, go to the Download page on our website and download a newer dongle driver version (Sentinel HASP Hardlock driver).

License file

License and licensee data are saved in a text file. In the standard case, this text file is bound to the MAC address of a computer. However, you can avoid that binding by concluding a license contract with INOSOFT GmbH. Notes on installation of the license file are included with the delivery.

Attention:
You can open the license file in a standard text editor and check the contents to see if the file belongs to you. You must neither change the contents nor save the changes. This will cause irreversible damage to the file. A damaged license file is not accepted by the system. To prevent this, always save a back-up copy of the license file.

Demo mode

You can install VisiWin for free to try it out and explore the functions.

IDE

In demo mode, you can only create and edit a project named `VW7Test`. In that project, you may use all functions. However, saving the project under a new name is not possible.

Runtime

Usually, an HMI project communicates with a controller, cyclically fetching data from the controller and displaying the data on a monitor. Without a runtime license, communication stops after one hour after HMI start. You can still use the HMI but no data will be received from or sent to the PLC. If you exit and restart the HMI, the communication will stop after one hour again.

3 System requirements

3.1 Operating systems and hardware requirements

The system requirements depend on the intended function and the operating system. To avoid performance losses, please observe the technical data below.

3.1.1 Operating systems

VisiWin 7.x supports the following operating systems:

Operating system	VisiWin 7 IDE	VisiWin 7 Runtime
Microsoft Windows		
Windows 7 (Professional/Enterprise/Ultimate) (32-bit/64-bit)	Smart Professional	Modern UI Client Classic UI Client Server
Windows 8.1 (Professional/Enterprise/Ultimate) (32-bit/64-bit)	Smart Professional	Modern UI Client Classic UI Client Server
Windows 10 (Pro/Enterprise) (32-bit/64-bit)	Smart Professional	Modern UI Client Classic UI Client Web UI Client Server
Windows 11 (Pro/Enterprise) (64-bit)	Smart Professional	Modern UI Client Classic UI Client Web UI Client Server
Microsoft Windows Embedded		
Windows Embedded Standard 7	Smart	Modern UI Client Classic UI Client Server

Operating system	VisiWin 7 IDE	VisiWin 7 Runtime
Windows Embedded Standard 8	Smart	Modern UI Client Classic UI Client Server
Windows 10 IoT Enterprise	Smart	Modern UI Client Classic UI Client Web UI Client Server
Microsoft Windows Server		
Windows Server 2008 R2 (64-bit) Windows Server 2012 (64-bit) Windows Server 2012 R2 (64-bit) Windows Server 2016 (64-bit)	Smart Professional	Modern UI Client Classic UI Client Server
Windows Server 2019 (64-bit) Windows Server 2022 (64-bit)	Smart Professional	Modern UI Client Classic UI Client Web UI Client Server

We recommend using the latest service packs.

3.1.2 Hardware requirements of the VisiWin 7 Smart IDE

Category	Minimum	Recommended
	with Windows 7 SP1	with Windows 10
Processor	1.6 GHz (dual core)	At least 2.2 GHz (dual core)
Main memory	2 GB	At least 4 GB
Minimum resolution	1366x768 pixels	At least 1920x1080 pixels
DirectX Hardware Acceleration	DirectX-9	At least DirectX-9

Category	Minimum	Recommended
Free HD memory for installation	1.5 GB	1.5 GB

Important:
Free HD memory for installation

To install and run VisiWin 7, the following software products are required:

Microsoft .NET Framework 4.0, Microsoft SQL Server Compact, Microsoft Windows Installer 3.1, Microsoft Visual C++ 2008 Redistributable, Sentinel Hasp Dongle Driver

These products are installed automatically during VisiWin 7 installation if they are missing. Additional HD memory will be needed in that case.

3.1.3 Hardware requirements of the VisiWin 7 Professional IDE

The hardware requirements of the VisiWin 7 Professional IDE are determined by the Visual Studio version in use. For more information, see Visual Studio documentation. For a list of Visual Studio versions supported by VisiWin 7, see Supported Visual Studio and .NET Framework versions (p. 9).

Category	Minimum	Recommended
	with Windows 7 SP1 and Visual Studio 2010	with Windows 10 and Visual Studio 2015
Processor	1.6 GHz (dual core)	At least 2.2 GHz (dual core)
Main memory	2 GB	At least 4 GB
Minimum resolution	1366x768 pixels	At least 1920x1080 pixels
DirectX Hardware Acceleration	DirectX-9	At least DirectX-9
Free HD memory for installation	1.5 GB	1.5 GB

Important:
Free HD memory for installation

To install and run VisiWin 7, the following software products are required:

Microsoft .NET Framework 4.0, Microsoft SQL Server Compact, Microsoft Windows Installer 3.1, Microsoft Visual C++ 2008 Redistributable, Sentinel Hasp Dongle Driver

These products are installed automatically during VisiWin 7 installation if they are missing. Additional HD memory will be needed in that case.

3.1.4 Hardware requirements of the VisiWin 7 runtime system

Category	Minimum	Recommended
	with Windows 7 SP1	with Windows 10
Processor	1.3 GHz (dual core)	At least 2.2 GHz (dual core)
Main memory	1 GB	At least 2 GB
Minimum resolution	800x600 pixels	At least 1366x768 pixels
DirectX Hardware Acceleration	DirectX-9	DirectX-9
Free HD memory for installation	100 MB	100 MB

Important:
Free HD memory for installation

To install and run VisiWin 7, the following software products are required:

Microsoft .NET Framework 4.0, Microsoft SQL Server Compact, Microsoft Windows Installer 3.1, Microsoft Visual C++ 2008 Redistributable, Sentinel Hasp Dongle Driver

These products are installed automatically during VisiWin 7 installation if they are missing. Additional HD memory will be needed in that case.

3.2 Supported Visual Studio and .NET Framework versions

VisiWin version	Visual Studio	.NET Standard Framework
7.0	Visual Studio 2010 or later	4.0 or later
7.1	Visual Studio 2013 or later	4.5 or later
7.2	Visual Studio 2013 or later	4.6.1 or later
7.3	Visual Studio 2019 or later	4.8

4 Installation

Since VisiWin 7 version 2017-1, there is only one installation program, which includes all current product versions. The installation program (setup) always includes the most recent VisiWin 7 IDE version and the current versions of all available runtime systems that can be installed concurrently. VisiWin extensions for Visual Studio can be installed optionally.

Example:

For example, installation version 2021-2 includes the following components:

- Current VisiWin 7 IDE version
- Runtime RT7.3 RC1 20220221.1
- Runtime RT7.2 SP15 20220218.1
- Runtime RT7.1 SP1 20190325.1
- Runtime RT7.0 SP6 20180514.1
- Visual Studio extensions

Note:
For more information on the current VisiWin version designations, please read the VisiWin Version Designations PDF. This document is available after registration and login at inosoft.com under **Downloads > VisiWin versions**.

VisiWin is no longer provided on CD but can be downloaded after registration and login from our website under **Download > VisiWin versions**.

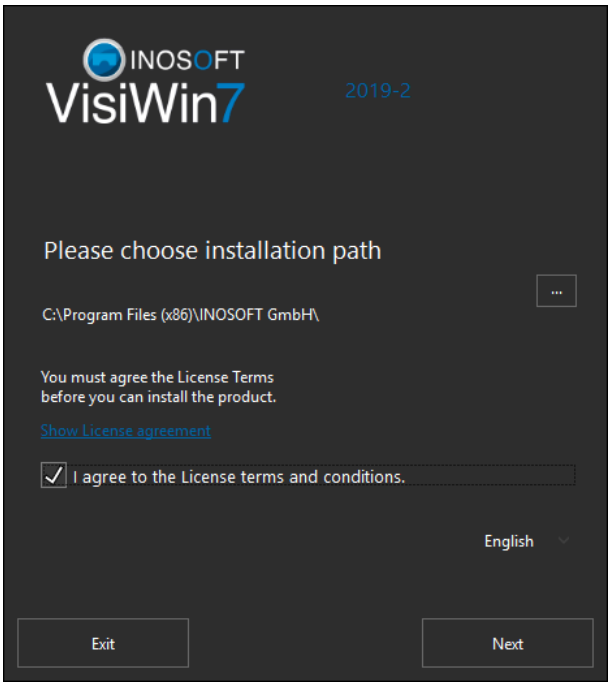
To download the VisiWin setup:

1. Sign up with inosoft.com and log in to the website using your username and password.
2. Navigate to the **Download > VisiWin version** section and click on the setup you want to install (e.g. **VisiWin 7 Setup 2021-2**) to start download. After download, the file (e.g. `VisiWin7_2021-2_20220228.2.zip`) is found in the `Downloads` folder by default.
3. Right-click on the ZIP file and select **Extract All...** from the context menu. You will find the setup (e.g. `VisiWin7.Setup.exe`) in the extracted folder.

To install VisiWin 7:

1. Double-click the file `VisiWin7.Setup.exe` to start setup.
2. Select the installation language and the installation path. You must agree to the license terms in order to proceed with the installation.

Figure 1. Select the installation path



3. Select the installation type.
    - **Development machine**

      Installs all components needed for development under VisiWin 7, such as the VisiWin 7 IDE, VisiWin 7 Visual Studio extensions (if Visual Studio is installed) and at least one version of the available runtime systems. You can select the runtime versions in the next dialog.
    - **Runtime machine**

      Installs only the components needed for the runtime system. You can select the runtime versions in the next dialog.
    - **Custom**

      Select this option if you want to select the components to be installed individually. This option is only recommended to experienced users.

Figure 2. Select the installation type

4. Select the runtime version you want to install.
   The appearance of the dialog depends on whether you selected development machine or runtime machine in the previous dialog.

Figure 3. Select runtime version for development machine

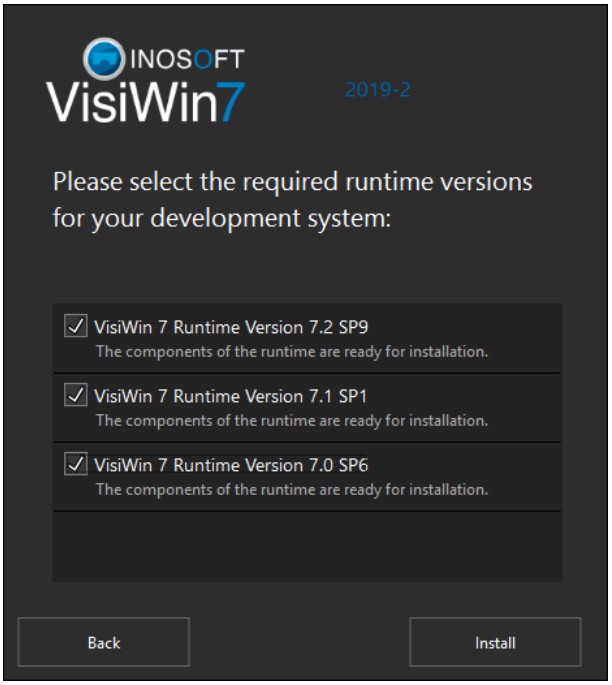
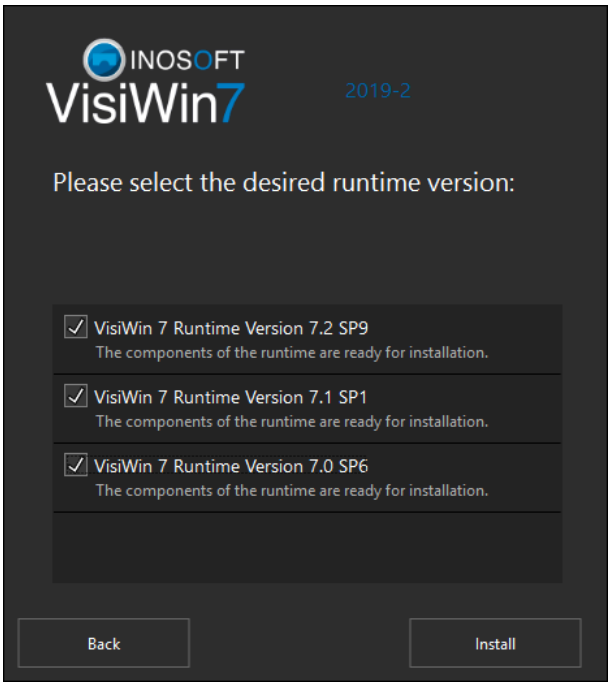


Figure 4. Select runtime version for runtime machine

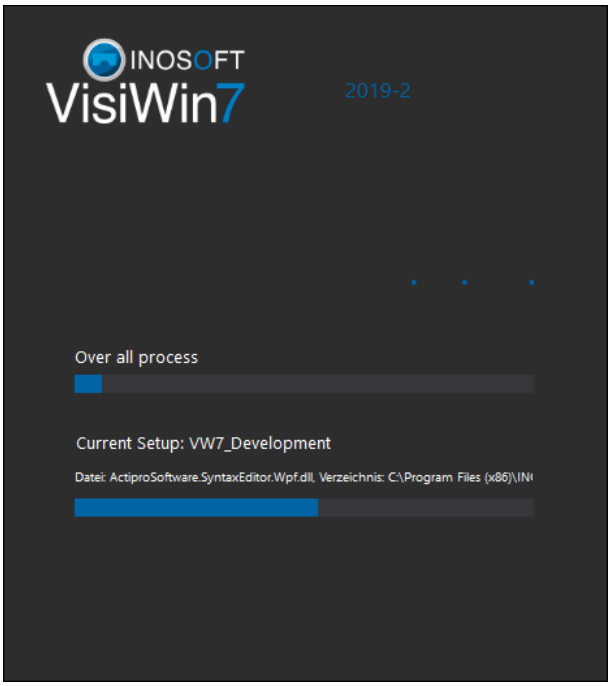


Note:

- You can select several versions at once. The runtime versions will be installed concurrently.
- If a runtime version is already installed, that option will be grayed out and a corresponding message will appear.
- If one of the selected runtime versions is already installed and the current installation program includes an update (a service pack, for example), that runtime version will be updated on the installation computer. Service packs cannot be installed concurrently.

5. Click on **Install**.
   The installation of the selected components has started. The installation progress is indicated by a progress bar.

Figure 5. Installation in progress



6. Finish your installation.
    - Click on **Close** to finish the installation without launching VisiWin 7.
    - Click on **Start VisiWin** to finish the installation and to launch VisiWin 7.

> **Note:**
> If errors occur during the installation, the window will show an error message. Please contact your VisiWin support in that case.

5 Quick start

The tutorials in this chapter will guide you through the individual stages of project development with VisiWin 7 Modern UI. You will learn the basics of how to work with VisiWin.

As the contents of this quick start guide build on one another, we recommend to work through the tutorials from the first to the last in consecutive order.

This guide covers the following project development stages:

My first project (p. 15)

Create your first VisiWin 7 project.

Variables (p. 22)

Set up communication with a (simulated) PLC.

Language switching (p. 28)

Define user interface texts that switch to another language at the push of a button.

Alarms (p. 31)

Define alarms you want to display in the application.

User management (p. 38)

Define users and add a predefined log-on dialog to your application.

Recipes (p. 44)

Define a recipe, create input controls for recipe values and add a predefined view that allows to load and save recipes.

Trends (p. 48)

Display curves of analog values.

5.1 My first project

After the successful installation of VisiWin 7, you can now start developing your first application.

In this tutorial, you will learn:

- how to create a new project in the IDE,
- how to start and terminate a project,
- which user interface components are already included in a new project,
- how to add a new view to your application.

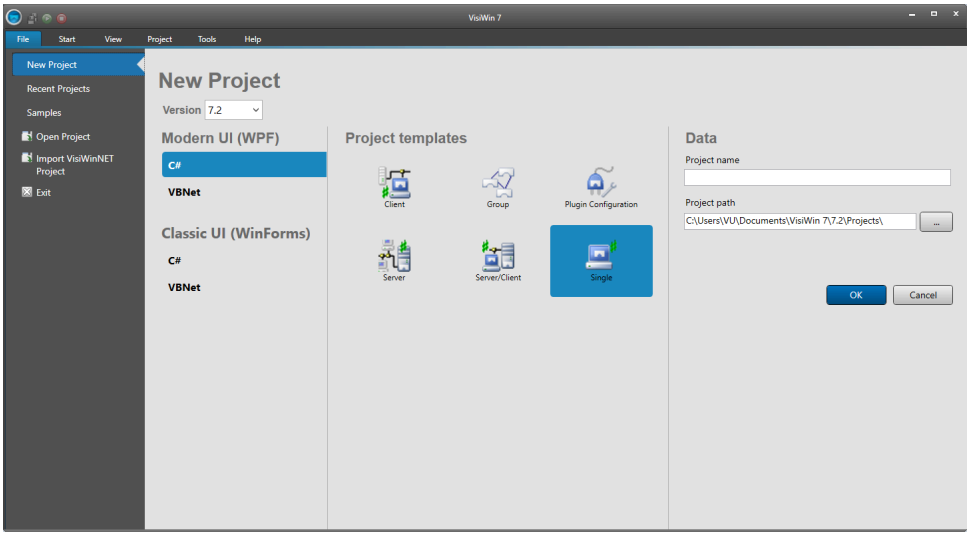
5.1.1 Create a new project

In this example, you will create a single-computer application with a WPF user interface (Modern UI) for a standard desktop PC and the programming language C#.

1. Start VisiWin 7 from the Windows start menu.

   When started, the IDE shows page **New Project** from the **File** menu.

   Figure 6. **New Project** dialog



2. Under **Modern UI (WPF)**, select **C#**.
3. Under **Project templates**, select **Single**.
4. In the **Project name** box, type `VW7Test`.

   The project directory is indicated under **Project path**. You can change the path if required.

   > **Important:**
   > In demo mode, you can only create and edit a project named `VW7Test`.

5. Click **OK** to confirm.

The project is created in the directory indicated by **Project path** and opened in the IDE in the **Start** menu.

5.1.2 Start and terminate project

The newly created project already includes an executable user interface. This means that you can start and test your project immediately after creation.

To start the project:

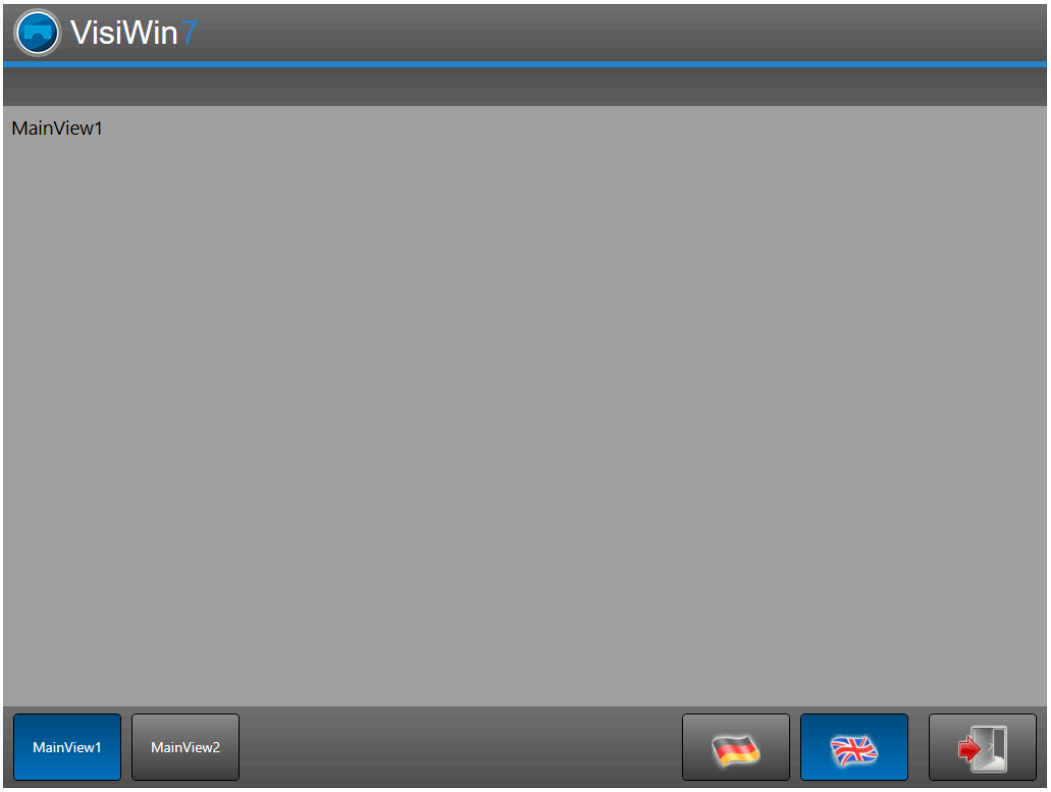
1. In the VisiWin ribbon, select **Project > Start** or press **F5**.

The project is started (compiled). The **Output** window shows the following message: `Starting project '<name>'`. The splash screen appears.

Figure 7. Splash screen

After a few seconds the application main window appears.

Figure 8. Main window



If you click on button **MainView1** and **MainView2**, the corresponding view appears (MainView1 or MainView2—see label at the top left corner). The header with the logo and the footer remain static. This *application frame* is the basic application you are going to develop further with the help of the instructions of this quick guide.

> **Note:**
> The buttons with the German and British flags switch the user interface language. Language switching will be described in detail later.

To understand the inner workings of the application frame, first terminate the project and return to the IDE.

To terminate the project:

- In the application window, click on  .

The application window is now closed. The **Output** window shows the following message: `Project '<name>' has exited.`

5.1.3 The application frame

Now have a look at your project in the IDE.

On the left-hand side of the program window, you can see the **Project Explorer** window. The project explorer shows a folder structure that allows to access all project areas. You can navigate in the project explorer as you would do in a file explorer, *expanding* folders (also referred to as *nodes*) to show the folder contents and *collapsing* nodes to hide the contents.

This topic deals with the user interface of your new project. The **Design** node provides access to the user interface of the project. This node includes all files that describe the user interface.

To understand how these files relate to each other, follow the steps below:

1. In the project explorer, expand node **Design > Windows**.

   You will find two files here: `MainWindow.xaml` and `SplashWindow.xaml`.

   `SplashWindow.xaml` defines the splash screen that appears when the application is started.

   `MainWindow.xaml` defines the application main window.

2. Now have a closer look at the *MainWindow*:
   a. In the project explorer, double-click on node **MainWindow.xaml**.

      The file is opened in the workspace, in the so-called *designer*.

At the bottom left corner of the IDE, you can see the **Document Outline** window. The document outline shows the structure of the file currently opened in the designer—in this particular case, the structure of `MainWindow.xaml`.

If you look at the structure in the document outline, you can see that MainWindow contains several *Regions*, i. a. **MainRegion**, **HeaderRegion**, and **FooterRegion**.

b. Select these regions in the **Document Outline** window one after the other.

The region selected in the document outline is automatically selected in the designer as well.

Regions divide the MainWindow into areas that display *views* (individual screen pages) at runtime.

3. Now have a look at the views of the application:
   a. In the project explorer, expand node **Design > Views**.

   Here you can find the nodes **FooterRegion**, **HeaderRegion**, and **MainRegion**. These nodes contain the views loaded into the respective regions of MainWindow.

   b. Expand node **FooterRegion** and open view **FooterView.xaml** in the designer.

   `FooterView.xaml` describes the footer of the application. The view displays two navigation buttons, two language switching buttons, and the application end button.

   c. Open the views contained in the nodes **HeaderRegion** and **MainRegion** and have a look at them in the designer.

It remains to be clarified how the application “knows” which view to load into which region of MainWindow.

4. The view to be loaded into a region is determined by the StartView property:
   a. In the designer, click on tab **MainWindow.xaml** in order to edit the view.

   > **Tip:**
   > A document once opened in the designer remains open until you close the document tab.

   b. In the **Document Outline** window, select **MainRegion**.

   The **Properties** window on the right-hand side of the IDE shows the properties of the object selected in the designer or in the document outline—in this particular case, the properties of MainRegion.

   c. In the **Properties** window, expand the **VisiWin** category.

This category includes the StartView property. In the StartView property, MainView1 is given as value. This means that MainView1 is loaded into MainRegion of MainWindow at application start.

d. Select the **FooterRegion** and the **HeaderRegion** of MainWindow in the **Document Outline** window and check the value in the **StartView** property.

You are familiar with the structure of the application now. The application frame contains the MainWindow and several regions placed on that window. The regions divide the window into areas into which views and their contents are loaded.

As a next step, you will add your own contents to the application frame.

5.1.4 Add a view

Now that you have familiarized yourself with the application frame, you can start adding your own contents to the project. In this section, you will create a new view for the MainRegion and enable navigation to that view by means of a NavigationRadioButton.

1. Add a new view to the project:
   a. In the project explorer, expand node **Design > Views**.
   b. Select node **MainRegion**.
   c. Right-click and select **Add new Element...** from the context menu.

      The **Add new Item** dialog appears.

   d. Select folder **Common** on the left-hand side of the dialog.

      The **View** template for new views appears on the right-hand side of the dialog.

   e. Under **Filename**, type a name for the new view , e. g. `MainView3`.
   f. Click on **Add** or press **Enter**.

   A new view has been added to the project. The view appears in the project explorer under the **MainRegion** node and is automatically opened in the designer.

2. Place a Label control on the view to show a text on the view:
   a. In the **Start** menu, select the **Label** control from the **Controls** group.
   b. Draw a rectangle of the desired size on the view in the designer using your mouse.

      The control has been generated on the view and is automatically selected in the **Document Outline** window.

   c. You can use your mouse to move the control in the designer if you want to change its position.
3. Determine the text you want to be displayed by the control by means of the Text property:

a. In the **Properties** window, expand the **Common Properties** category.
b. Determine the desired text in the **Text** property, e. g. `Welcome`.
c. Press **Enter** to confirm.

The text appears on the view in the designer.

4. Place a new NavigationRadioButton on the FooterView to enable the user to navigate to the view:
   a. In the project explorer, expand node **FooterRegion**.
   b. Open the **FooterView.xaml** in the designer.
   c. Copy the button labeled **MainView2**:
      i. Click on the button to select it.
      ii. Press **Ctrl+C**.
      iii. Press **Ctrl+V**.

   A third NavigationRadioButton appears on the view in the designer and in the **Document Outline** window. However, there is no margin between the new button and the button to the left. To set the margin, use the Margin property.

5. Set the margin in the Margin property:
   a. In the **Properties** window, expand the **Layout** category.
   b. In the **Margin** property, type `10` (pixels) in the box for the left margin (arrow left).
   c. Press **Enter**.

> **Tip:**
> The **Properties** window shows a search box that you can use to find the desired property more quickly. To see all properties again, delete the search box contents.

6. Type a label text in the **Text** property, e. g. `MainView3`.
7. Select your new view (`MainView3`) from the drop-down menu in the **ViewName** property.

   The view will appear in the MainRegion (this is already determined by the **RegionName** property) when the user clicks on the newly added **MainView3** button.

Start the project to test the application.

Your application now contains three navigation buttons. If you click on the third navigation button, the application shows your newly added view with your text on it.

Terminate the application.

5.2 Variables

In the previous tutorial, you created your first Modern UI project with VisiWin 7, familiarized yourself with the project's application frame and carried out minor modifications.

Your application has a functioning user interface now but cannot display machine data or pass user input on to the machine. To enable this, you have to set up a *communication channel* and determine the *variables* via which data will be exchanged with the controller. All this is defined in the *VisiWin Variables system module*. In the IDE, you can access the system module via project explorer node **Variables**.

In this tutorial, you will learn:

- how to create a communication channel,
- how to import process variables via the channel into the project,
- how to edit variables in the IDE,
- how to bind a variable to a control in order to show the variable value in the application.

5.2.1 Create a communication channel

A communication channel determines the communication component via which data will be exchanged. *Communication components* are device drivers that support the specific protocol of a PLC and exchange process data with the HMI.

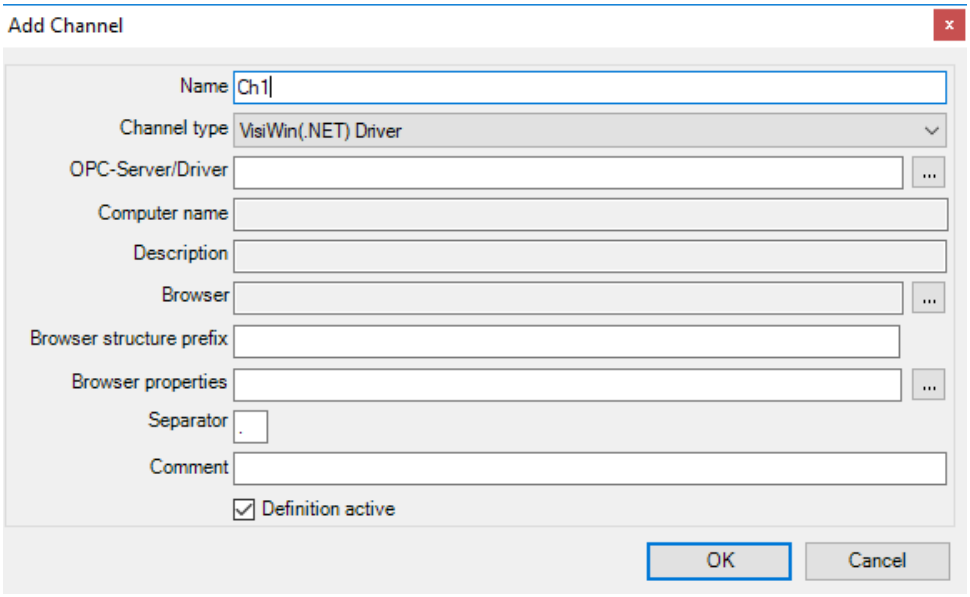
Note:
You do not need a "real" communication component and a PLC to be able to follow the instructions in this tutorial. Instead, you will use the sample driver included in the VisiWin 7 installation that simulates PLC data exchange.

Create a new communication channel:

1. In the project explorer, expand node **Variables**.
2. Select node **Channels** below.
3. Right-click and select **New** from the context menu.

   The **Add Channel** dialog appears.

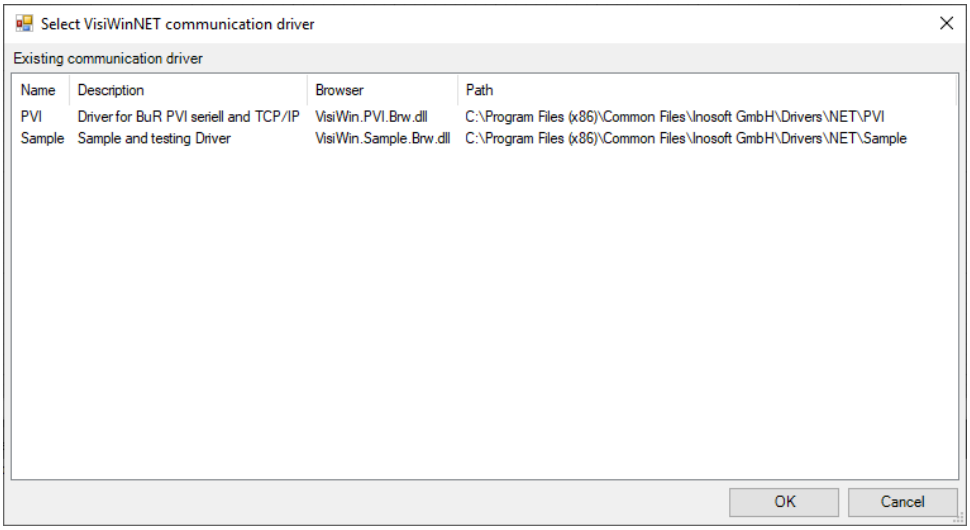
Figure 9. **Add Channel** dialog



4. Ensure that **VisiWin(.NET) Driver** is selected in the **Channel type** box.
5. Click on the ... button next to the **OPC-Server/Driver** box.

   The **Select VisiWinNET communication driver** dialog appears. This dialog shows a list of all installed drivers.

Figure 10. **Select VisiWinNET communication driver** dialog



6. Select the driver named **Sample**.

7. Click **OK** in both dialogs.

> **Note:**
> If you need a specific driver, please send a request to INOSOFT. Once the driver is installed, you can select it in the dialog.

A new communication channel has been added to the project. A new entry **Ch1:Sample** appears under the **Channels** node. This entry includes the channel name (Ch1) and indicates the communication component for the channel—in this case, the name of the sample driver (Sample).

If you select the channel in the project explorer, the **Property Pages** window shows the channel *parameters*. You can configure the communication channel using these parameters.

5.2.2 Import process variables into the project

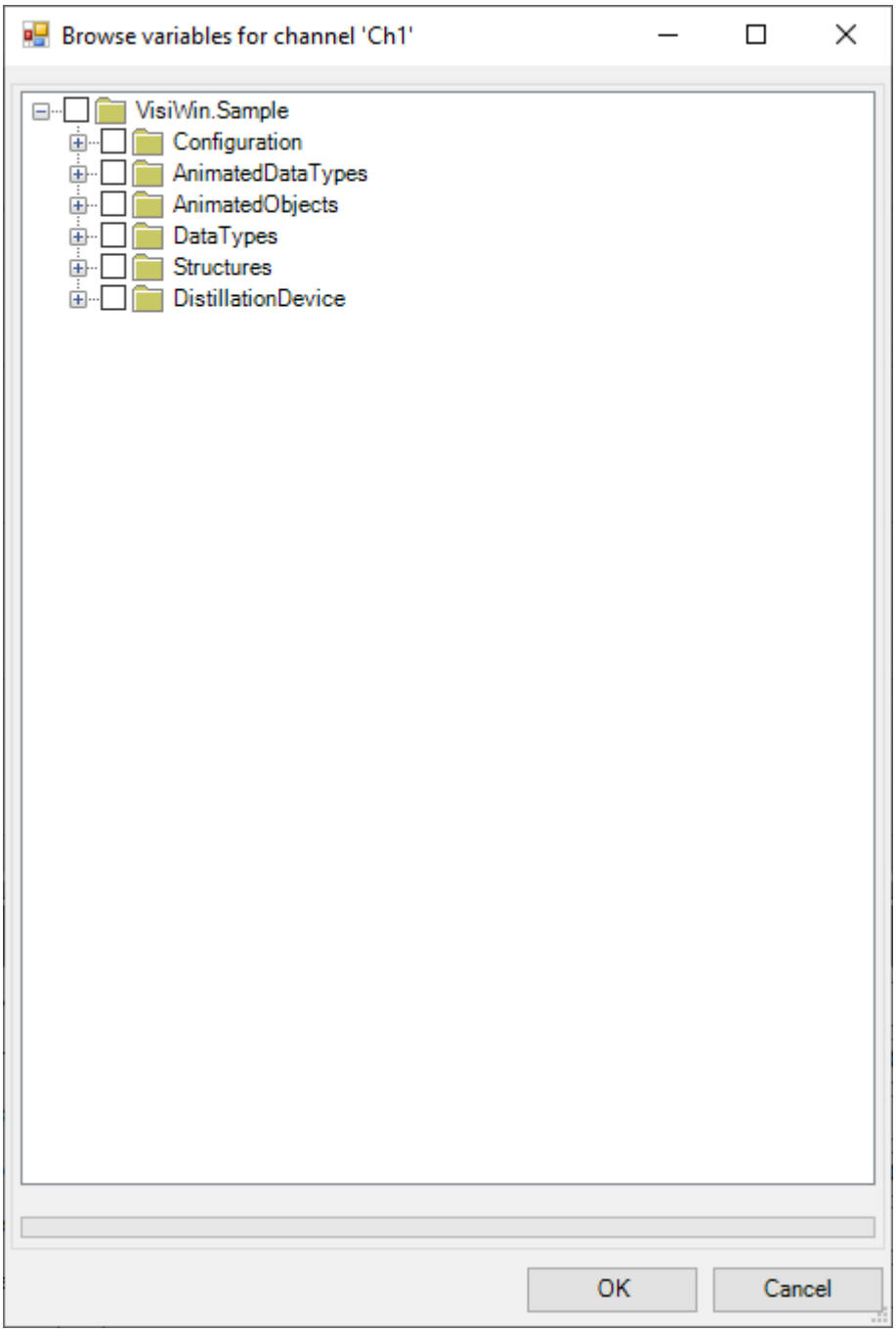
You have defined a communication channel and specified the communication component for your project. Now you can import process variables from the communication component into your project.

To import process variables into your project:

1. In the project explorer, select the node of the newly created channel.
2. Right-click and select **Browse Variables...** from the context menu.

   The **Browse variables for channel '<channel name>'** dialog appears.

Figure 11. **Browse variables for channel '<channel name>'** dialog



The dialog shows the variables of the communication component as a tree structure. The branches of the tree represent *namespaces* whereas the ends represent the actual variables that can be selected.

3. Put a tick into the box in front of the **VisiWin.Sample** root node to select the node.

   This selects all variables known to the driver.

4. Click **OK** to confirm.

The variable definitions have been imported into the project. If you expand the channel node in the project explorer, you can see the same namespace structure as in the Select Variables dialog.

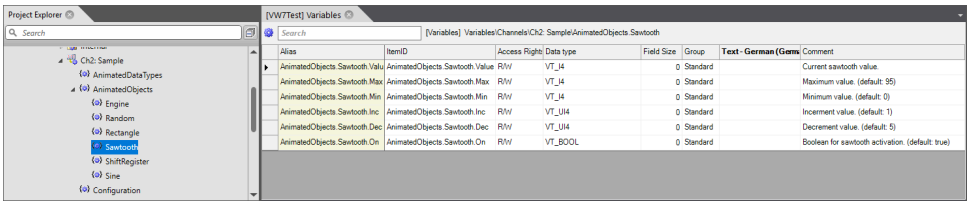
5.2.3 Show and edit variable definitions

The imported variable definitions are displayed in the *table editor* in the workspace if you double-click on a namespace node.

To open and edit a process variable in the table editor:

1. In the project explorer, expand node **Variables > Ch1:Sample > AnimatedObjects**.
2. Double-click on namespace **Sawtooth** to open the process variable definitions of that namespace.

Figure 12. Table editor

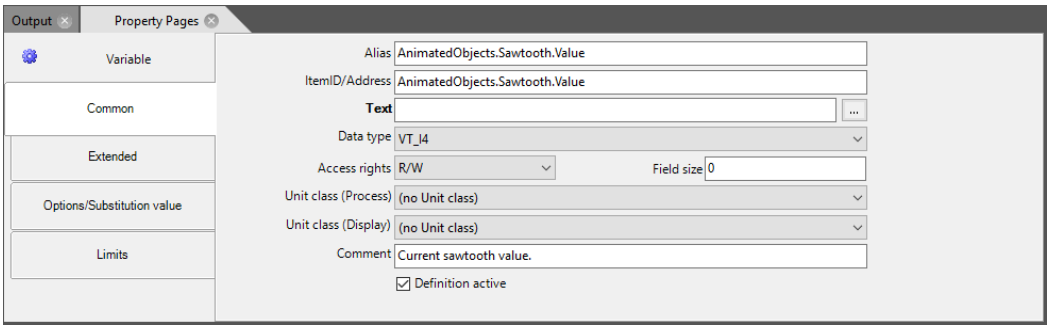


The table editor shows the variable definitions of the selected namespace in a table. A row represents a variable definition. The columns contain some of the parameters of the variable definition.

Tip:
Use the **Options** dialog to select a different set of parameters you want to show in the table editor. To open the dialog, right-click on the table editor and select **Options** from the context menu.

The **Property Pages** window shows all parameters of a variable definition if you select the row of the variable definition in the table editor.

Figure 13. Property pages



5.2.4 Bind a variable to a control

Having imported several process variables into a communication channel of your project, you can now begin working on the actual visualization task. For example, you can put a NumericVarOut control on a view and bind a variable to the control in order to show the variable value in an output field in the running application.

To bind a process variable to a control:

1. Open **MainView1.xaml** in the designer.
2. Place a **NumericVarOut** control on the view.

   The control has been generated on the view and is automatically selected.

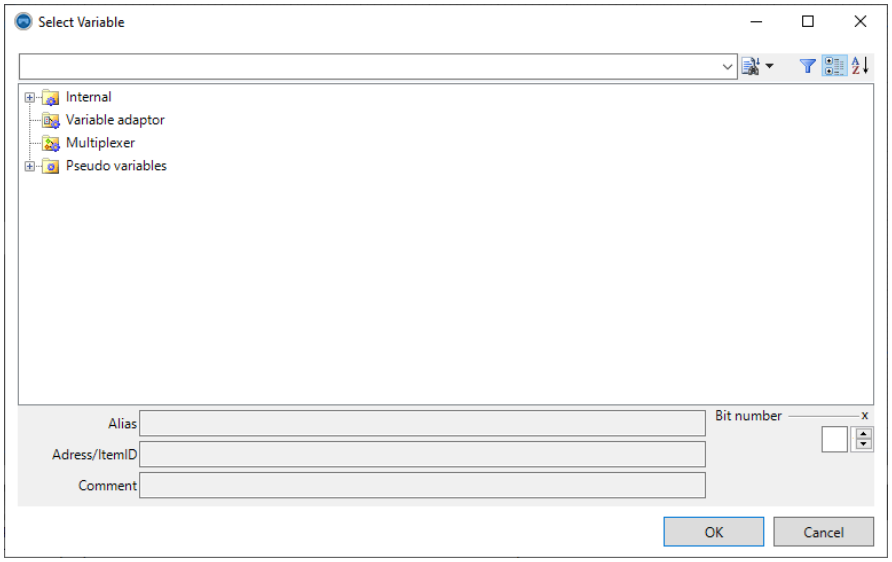
3. In the VariableName property, select the variable whose values you want to show in the control:
   a. In the **Properties** window, expand the **Variable** category.

      This category contains the **VariableName** property.

   b. Click on the button next to the **VariableName** property.

      The **Select Variable** dialog appears.

      Figure 14. **Select Variable** dialog

   c. Expand node **Ch1 > AnimatedObjects > Sawtooth** and select the *Value* variable.

   d. Click **OK** to confirm.

Tip:
If you need several controls of the same type and with the same settings but want to bind them to different variables, there is a convenient way to duplicate the control.

1. Right-click on the control on the view in the designer.
2. Select **Duplicate with variable selection** from the context menu.

   The **Select Variable** dialog appears.
3. Select the process variables the values of which you want to display in the controls of the same type as the selected control.
4. Click **OK** to confirm.

Start the project to test the application.

The output field on **MainView1** shows the constantly changing value of the simulated process variable.

Terminate the application.

5.3 Language switching

An HMI is mostly distributed internationally and therefore needs to be localized, i. e. adapted to local markets. The operator must be enabled to change the language of all texts of the application at the push of a button.

VisiWin allows you to create multilingual texts. Multilingual texts (referred to as *localizable texts* in VisiWin) are provided and managed by the *VisiWin Language Switching* system module. In the IDE, you can access the system module via the **Language switching** node in the project explorer. VisiWin includes the *LanguageChangeRadioButton* control that allows to change the language at runtime.

In this tutorial, you will learn:

- how to create a localizable text,
- how to bind a localizable text to a control in order to show the text in the control,
- how to use the LanguageChangeRadioButton in order to change the language of the application.

5.3.1 Create a localizable text

If you expand the **Language switching** node in the project explorer, you can see that the node already includes several subnodes.

The **Configuration** node allows you to specify the languages for your application. You can see the languages in the table editor if you double-click on the **Languages** node. This is where you can add

new languages to your project and remove existing languages. The application frame of your project is already configured for German and English.

All other subnodes are so-called *text groups* containing the actual texts or other subgroups. As the application frame already contains texts, most of the text groups are not empty.

Add any texts that you create in your project to the **User texts** group.

To create a localizable text:

1. Create a new text group:
    a. In the project explorer, expand node **Language switching**.
    b. Select node **User texts**.
    c. Right-click and select **New** from the context menu.

    A new text group **Text Group1** has been created within the **User texts** group. The **Property Pages** window shows the parameters of the text group you have just created. The **Name** parameter is the only parameter in this case. The new text group is displayed in the table editor.
2. In the **Name** parameter, indicate a name for the new text group:
    a. In the **Property Pages** window, indicate a name in the **Name** parameter for the text group, e. g. `MainView3`.

    > **Tip:**
    > It is good practice to organize text groups by views where the texts are used.

    b. Click beyond the **Property Pages** window, e. g. on the workspace.

    The text group name adapts in the project explorer.
3. Create a text definition within the new text group:

    Right-click on the table editor and select **New** from the context menu.

    A new text definition has been generated and appears as a new row in the table editor.
4. Indicate a name for the text definition as well as the texts for the project languages:
    a. In the **Name** column in the table editor, type the name `WelcomeText`.
    b. In the **German (Germany) (1031)** column, type `Willkommen`.
    c. In the **English (United States) (1033)** column, type `Welcome`.

You have created a text group for the texts of MainView3 and defined a localizable text within that text group. This localizable text includes a text for German and one for English. Now you can bind

the localizable text to a control so that the control shows the right text in the selected user interface language.

5.3.2 Bind a localizable text to a control

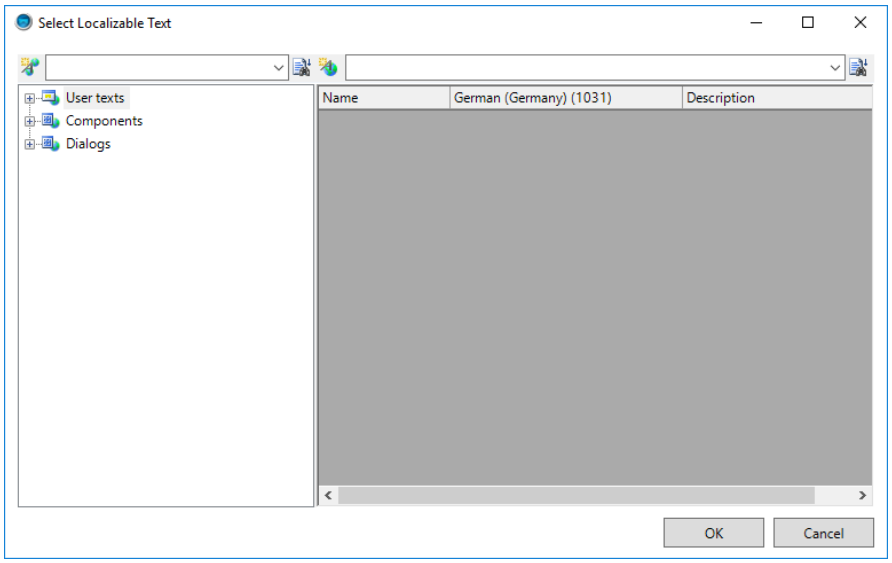
You have already added a VisiWin Label control to your application to which you have bound a text using the Text property (see Add a view (p. 20)). However, this is not a multilingual text. If you click on the button with the British or German flag, the text in the Label control will not change. To bind a localizable text to the control, you have to use the LocalizableText property.

To bind a localizable text to a control:

1. Open **MainView3.xaml** in the designer.
2. Select the **Label** control on the view.
3. Specify the text in the **LocalizableText** property:
    a. In the **Properties** window, expand the **Common Properties** category.
    b. Click on the  button next to the **LocalizableText** property.

    The **Select Localizable Text** dialog appears.

    Figure 15. **Select Localizable Text** dialog

    c. In the dialog, expand text group **User texts** and select text group **MainView3** previously created.

    Text definition `WelcomeText` appears on the right-hand side of the dialog.

    d. Click **OK** to confirm.

Start the project to test the application.

Navigate in your application to MainView3 by clicking on the **MainView3** button. The **Willkommen** text is displayed on the view. If you click on the button with the British flag, **Welcome** will appear on the view instead.

Terminate the application.

5.3.3 The LanguageChangeRadioButton

VisiWin includes a special control that allows to change the user interface language—the LanguageChangeRadioButton. The footer of the application frame already includes two controls of this type.

If you open the FooterView in the designer (**Design > Views > FooterRegion > FooterView.xaml**) and select a button with a flag on it, you can see in the **Document Outline** window that the control is a LanguageChangeRadioButton.

The reference to a language is made in the LCID property of the LanguageChangeRadioButton. The *LCID* (*locale identifier*) is a four-digit numerical value permanently assigned to a language. For example, the LCID for German (Germany) is 1031.

Select the LanguageChangeRadioButton with the German flag and expand the **Common Properties** category in the **Properties** window. The category contains the **LCID** property, which is set to `1031`. This means that all texts of the application switch to German when the operator clicks on the LanguageChangeRadioButton.

Note:
The VisiWin 7 installation includes a language switching sample project. To open the sample project, select **File > Samples > Language Switching > Open** in the IDE. Start the project and test the application at runtime. Terminate the application and take a closer look at the project in the IDE.

5.4 Alarms

A core task of any HMI is to collect, display and acknowledge *alarms* of various levels (alarms, messages, and hints) that occur in the machine.

The *VisiWin Alarms system module* allows you to define alarms, to select the alarm triggering process variables, to set colors and other display options and to add further information to alarms. In the IDE, you can access the system module via the **Alarms** node in the project explorer.

VisiWin includes two controls that display alarms: *AlarmLine* and *AlarmList*.

In this tutorial, you will learn:

- how to define alarms in the IDE,
- how to display alarms in an alarm line,
- how to display alarms in an alarm list,
- how to display only alarms of a particular category.

5.4.1 Create alarms

To be able to create an alarm, you need a PLC variable that will trigger the alarm in the HMI. Instead of a real process variable, you will create an internal variable (i. e. a variable that is defined in the VisiWin Variables system module but has no counterpart in a PLC), the value of which you can manipulate by means of input controls in the running application, thus deliberately triggering alarms for learning purposes.

1. Create a new variable in your project:
   a. In the project explorer, expand node **Variables > Channels**.
   b. Double-click on the **Internal** node.

      The table editor is now open in the workspace.

   c. Right-click on the table editor and select **New** from the context menu.
   d. Under **Alias**, type `AlarmSource` as a name for the new variable.

   A new variable of the VT_UI2 data type (value range: -32768 – +32767) has been created. You will use some of the 16 available bits in step 5 (p. 33).

   In VisiWin, each alarm must belong to an alarm group.

2. Create a new alarm group:
   a. Expand node **Alarms**.
   b. Select node **Alarm groups**.

   c. Right-click and select **New** from the context menu.

   A new alarm group named **Alarm group1** has been created; the table editor opens in the workspace. Now you can define individual alarms in the alarm group.

3. Create eight alarms in the alarm group:
   a. Click on the upper area of the table editor. This is where you can create alarm definitions. In the lower area, you can see and create alarm groups (subgroups) within the current alarm group.
   b. Press **F8** eight times.

   Eight new alarm definitions have been created and appear as new rows in the table editor.

4. Specify the alarm triggering variable (event variable) for each of the eight alarms:

a. In the first row, click on the cell in the **Event variable** column.

   The … button appears in the cell.

b. Click on the … button.

   The **Select Variable** dialog appears.

c. In the dialog, expand node **Internal**.

d. Select variable *AlarmSource*.

e. Click **OK** to confirm.

   The *AlarmSource* variable appears in the **Event variable** column of the first alarm definition.

f. Specify the same event variable in the other seven alarm definitions.

> **Tip:**
> Use the **Fill** function to copy a parameter value to all selected definitions:
> 1. Select the cell whose parameter value you want to copy to other definitions.
> 2. Expand the selection to include the cells which you want to copy the parameter value to.
> 3. Right-click and select **Fill** from the context menu or press **Ctrl+U**.

The *AlarmSource* variable is now indicated as the alarm triggering variable in each of the eight alarm definitions.

5. Specify the number of the alarm triggering bit in each of the eight alarm definitions:
   a. For *Alarm1*, specify `0` in the **Bit number** column.
   b. For alarms *Alarm2–Alarm8*, increase the **Bit number** by 1 for each alarm definition.

> **Tip:**
> Use the **Step 1** function to increase the parameter value by 1 for each of the selected definitions:
> 1. Select the cell whose parameter value you want to increase by 1 for other definitions.
> 2. Expand the selection to include the cells for which you want to increase the parameter value by 1.
> 3. Right-click and select **Step 1** from the context menu or press **Ctrl+1**.

The alarm triggering bit is now indicated in each of the eight alarm definitions.

In VisiWin, an *alarm class* must be indicated for each alarm. This is what you will do in the next step.

6. Indicate an alarm class in each alarm definition:
    a. For alarms 1–3, select **Alarm** in the **Alarm class** column.
    b. For alarms 4–6, select the **Message** alarm class.
    c. For alarms 7–8, select the **Hint** alarm class.

    Alarm classes Alarm, Hint, and Message have already been defined in the application frame. To view the alarm class definitions, double-click on node **Alarms > Alarm classes** in the project explorer.

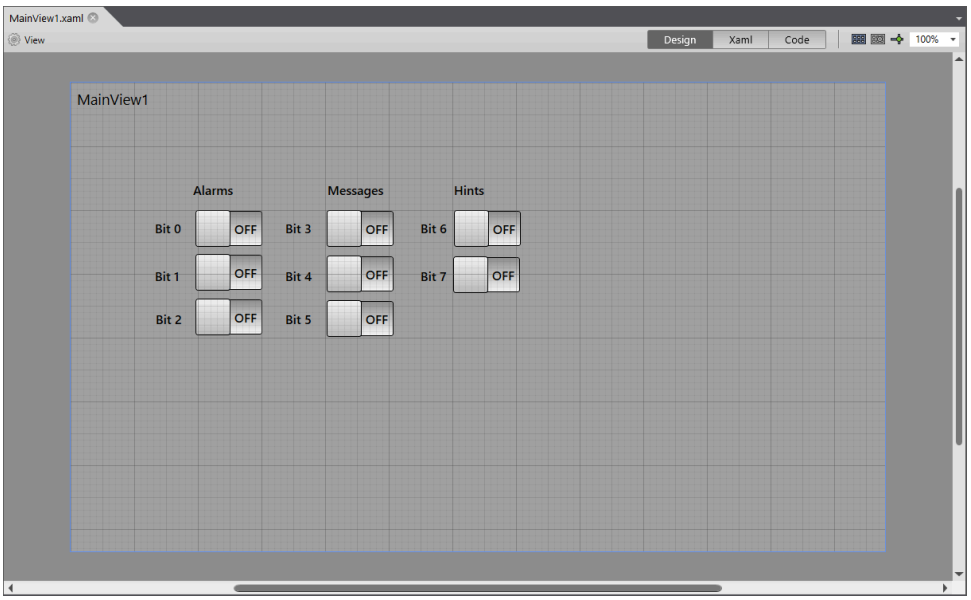
5.4.2 Display alarms in an alarm line

An AlarmLine control has already been implemented in the header (**HeaderView.xaml**) of the application frame. Alarms, too, have already been created in the project. To test the alarm functionality, you have to trigger alarms manually. For example, you can use Switch controls to set individual alarm bits to trigger the corresponding alarms.

1. Add eight Switch controls for the eight alarms (**Start > Controls > Buttons > Switches and Buttons > Switch**) to **MainView1.xaml**.

    You can place the controls in a row or in three columns grouped by alarm class. You can also add labels (*Label* control) to the view for clarity.

    Figure 16. View with controls, using which alarms can be triggered manually



2. Specify the variable as well as the alarm triggering bit for the first Switch control:

a. In the **Variable** category, set the **VariableName** property to *AlarmSource*.
b. Set the **BitNumber** property to `0`.

3. For the second Switch control, select *AlarmSource* as triggering variable and bit `1` as triggering bit.
4. Select the alarm variable and the triggering bit for the other Switch controls, increasing the bit number by 1 each time.

Start the project to test the application.

If you set the bit by clicking on the button, the corresponding alarm appears in the alarm line.

The AlarmLine shows only one alarm at a time. An overview of all triggered alarms is provided by the AlarmList control, which you will add in the next step.

Terminate the application.

5.4.3 Display alarms in an alarm list

An alarm list occupies much more space than an alarm line. Therefore, a whole view is required for the alarm list.

1. Add an **AlarmList** control to (**Start > Controls > System > Alarms > AlarmList**) to the **MainView2.xaml** view.
2. Specify the position and size of the control using the properties of the **Layout** category:

Property	Value
Width	Auto
Height	Auto
HorizontalAlignment	Stretch
VerticalAlignment	Stretch
Margin	0, 0, 0, 0

These settings stretch the control across the entire view.

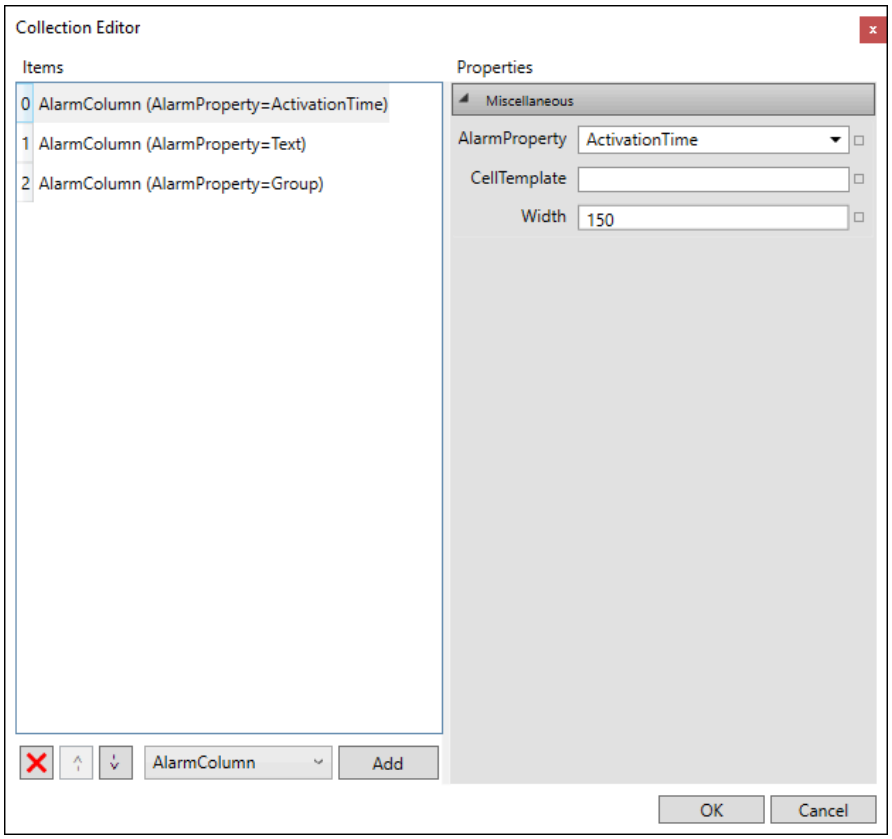
3. Use the **Columns** property to add another column that will display the alarm class:
   a. In the **Properties** window, expand the **Alarm** category.

      This category includes the **Columns** property.

   b. Click on the … button.

      The **Collection Editor** dialog appears.

Figure 17. **Collection Editor** dialog



c. Click on the **Add** button.

   A new *AlarmColumn* element has been added to the **Items** list. In the **AlarmProperty** parameter, **AcknowledgeTime** is indicated. This means that the new column will display the time when the alarm was acknowledged. To display the alarm class instead, you have to set the **AlarmProperty** to **Class**.

d. Click **OK** to confirm.

The control now displays the **Class** column.

You can add columns to the AlarmLine in the same way. If the columns you added are not visible in the control, reduce the width of the existing columns using the **Width** property.

Start the project to test the application.

Trigger one or more alarms and click on the **MainView2** button to navigate to MainView2. The AlarmList shows the triggered alarms.

Terminate the application.

5.4.4 Filter alarms

Not every alarm requires immediate action of the operator. Messages and hints may require acknowledgment but not instantly. You can filter active alarms so that only major errors will call for attention.

To display only alarms of the **Alarm** class in the AlarmLine:

1. Select the **AlarmLine** control in the designer or in the **Document Outline** window.
2. In the **Properties** window, expand the **AlarmFilter** category.

   This category includes the **DesiredClasses** property.
3. In the **DesiredClasses** property, indicate the name of the class that you want to display in the AlarmLine, e. g. `Alarm`.
4. Press **Enter**.

Start the project to test the application.

If you trigger alarms 1–3, the corresponding alarm is displayed in the AlarmLine.

Now, trigger alarms 4–8. These alarms are not displayed in the AlarmLine since they do not belong to the Alarm class that you selected in the DesiredClasses property of the AlarmLine control.

Navigate to the AlarmList. The AlarmList shows all triggered alarms since the DesiredClasses property was not set for the AlarmList.

Example:
Filtering alarms in alarm lists is useful, for example, if your application includes a large number of alarms. You could add several alarm lists to the alarm overview, where each of these lists would show only alarms of a particular group (DesiredGroups property).

Terminate the application.

Tip:
The VisiWin 7 installation includes an alarms sample project. To open the sample project, select **File > Samples > Alarms > Open** in the IDE. Start the project and test the application at runtime. Terminate the application and take a closer look at the project in the IDE.

5.5 User management

The VisiWin user management allows to define user access rights for the HMI application. Input controls can be locked and access to information displayed by output controls denied to particular user groups, thus ensuring that only authorized personnel have access to specific functional components.

The *VisiWin User Management system module* (project explorer node **User management**) allows you to define *user groups* such as `operator`, `service technician`, or `administrator`. Within the user groups, you have to define individual *users*. *Rights*, e. g. `may operate machine`, `may terminate application`, `may reset service intervals`, `may manage users`, etc., are allocated to groups, not to individual users. Users only include credentials such as name, password, and personalized codes.

In this tutorial, you will learn:

- how to create user groups and users,
- how to create rights,
- how to bind rights to controls,
- how to add the available login dialog to the project,
- how to add the available user administration dialogs to the project.

5.5.1 Create user groups and users

Similar to the Language Switching system module, where texts are created within text groups, and the Alarms system module with its alarm groups and alarms, the VisiWin User Management system module, too, includes user groups and users. Every user must belong to a user group.

1. Create a new user group named `Operator`:
    a. In the project explorer, expand node **User management**.
    b. Double-click on the **User groups** node.

    The table editor is now open in the workspace. As you have not defined any user groups yet, the table editor is empty.

    c. Click on the table editor once and press **F8**.

A new user group **Group1** has been created. The **Property Pages** window shows the parameters of the user group you have just created.

d. In the **Name** parameter, type `Operator`. This is the name for the first user group.

2. Create a second user group named `Service`.

a. Click on the table editor once and press **F8**.

b. In the **Name** parameter, type `Service`.

3. Create a user within the `Operator` user group:

a. In the project explorer, double-click on the **Operator** node.

b. Click on the table editor once and press **F8**.

c. Set the following parameters for the newly created user either in the table editor or in the **Property Pages** window:

Parameters	Value
Login	TestOperator
Full name	A. B.
Password	asdf
State	Activated

4. Create a second user within the `Service` user group with the following parameters:

Parameter	Value
Login	TestService
Full name	C. D.
Password	asdf
State	Activated

You have created two user groups with one user in each group. Now, you can define rights and allocate them to the user groups.

5.5.2 Create and allocate rights

Create rights and allocate them to the user groups.

1. Create a new right `ApplicationEnd` and allocate that right to user groups `Operator` and `Service`:
    a. In the project explorer, double-click on the **Rights** node within the **User management** node.

    The table editor is now open in the workspace. As you have not defined any right yet, the table editor is empty.

    b. Click on the table editor once and press **F8**.
    c. Either in the table editor or in the **Property Pages** window, type `ApplicationEnd` in the **Name** parameter to indicate a name for the new right.
    d. In the **User groups** parameter, select both `Operator` and `Service`.

    You have created a new right `ApplicationEnd` and allocated that right to user groups `Operator` and `Service`.
2. Create a new right `OperatorTasks` and allocate that right to user group `Operator`.
3. Create a new right `ServiceTasks` and allocate that right to user group `Service`.

You have created three rights: A common right for both user groups `Operator` and `Service` as well as a specific right for each user group. To unlock a control for a specific user group, you have to bind the right to the control using the AuthorizationRight property.

5.5.3 Bind rights to controls

You can bind exactly one right to a VisiWin control using the AuthorizationRight property. That control is only available to the user group that includes the right indicated in the AuthorizationRight property.

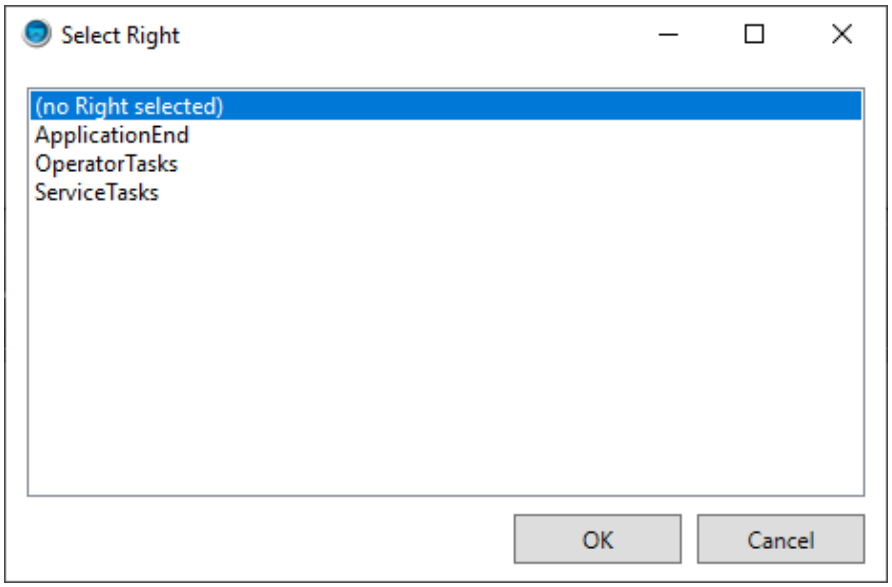
1. Open **FooterView.xaml** in the designer.
2. Select the **MainView1** button on the view.
3. Bind the `OperatorTasks` right to the **MainView1** control using the **AuthorizationRight** property.
    a. In the **Properties** window, expand the **Authorization** category.

    This category includes the AuthorizationRight property.

    b. Click on the  button next to the **AuthorizationRight** property.

    The **Select Right** dialog appears.

Figure 18. **Select Right** dialog



c. Select the `OperatorTasks` right in the dialog.

d. Click **OK** to confirm.

You have allowed the `Operator` user group to access the MainView1 by binding the `OperatorTasks` right to the NavigationRadioButton that displays the view.

4. Now bind `ServiceTasks` right to the **MainView2** button using the **AuthorizationRight** property.
5. Bind the `ApplicationEnd` right to the Exit button using the AuthorizationRight property.

You have explicitly allowed the `Operator` user group to access MainView1, thus implicitly denying access to the `Service` user group. MainView2, however, can be viewed only by user group `Service`. Both user groups may terminate the application since both have the right to exit the application that you have bound to the Exit button.

If you start the project, you will discover that buttons that have a right bound to them in the AuthorizationRight property are disabled. This is because you have not logged in as a user yet. VisiWin includes a predefined login dialog that you can add to your project.

5.5.4 Add the login dialog to the project

VisiWin includes the UserOverview template that allows users to log in to the system. The following steps show you how to integrate that template into your project.

1. Add the UserOverview template to the project:
   a. In the project explorer, expand node **Design > Views**.
   b. Select node **MainRegion**.
   c. Right-click and select **Add new Element...** from the context menu.

      The **Add new Item** dialog appears.

   d. Select folder **User** on the left-hand side of the dialog.

      The available templates are displayed on the right-hand side of the dialog.

   e. Select project template **UserOverview**.

   f. Press **Enter** to confirm.

   The **MainRegion** node now contains the UserOverview.xaml as well as a number of other associated templates such as the login dialog LogOnOff.xaml.

2. Bind the UserOverview.xaml to a NavigationRadioButton to enable users to navigate to the view:

   a. Open the **FooterView.xaml** in the designer.

   b. Select the NavigationRadioButton labeled with **MainView3**.

   c. Set the following control properties:

Property	Value
RegionName	MainRegion
ViewName	UserOverview
Text	User overview

3. Define the UserOverview as the first view to be displayed:
   a. Expand node **Windows** in the project explorer.
   b. Open **MainWindow.xaml** in the designer.
   c. In the **Document Outline** window, select the **MainRegion** element.
   d. In the **StartView** property, select **UserOverview**.

Start the project to test the application.

As soon as the application starts, the UserOverview ist displayed. Buttons **MainView1**, **MainView2** as well as the exit button are disabled since rights are bound to them.

Click on the **Log on/off** button on the UserOverview. A login dialog appears. Log in as user `TestOperator` using password `asdf`. After successful login, the **MainView1** and buttons are enabled. If you log in as `TestService`, the **MainView2** button is enabled while the **MainView1** button

is disabled since user group `Service` includes the `ServiceTasks` right. The  button remains enabled since both users have the `ApplicationEnd` right, which is included in their respective user group.

Terminate the application.

5.5.5 Add the user administration dialogs to the project

Users of an IT system are usually managed by an administrator. The UserManager template is a predefined user interface component that allows to administrate users.

1. Add the UserManager template to the project:
    a. Select node **MainRegion** in the project explorer.
    b. Right-click and select **Add new Element...** from the context menu.

    The **Add new Item** dialog appears.

    c. Select folder **User** on the left-hand side of the dialog.

    The available templates are displayed on the right-hand side of the dialog.

    d. Select the **UserManager** template.

    e. Press **Enter** to confirm.

    The **MainRegion** node now includes the UserManager.xaml.
2. Add a new NavigationRadioButton to the footer to enable users to navigate to the UserManager view:

    a. Open the **FooterView.xaml** in the designer.

    b. Add a **NavigationRadioButton** control to the view.

    c. Set the following control properties:

Property	Value
RegionName	MainRegion
ViewName	UserManager
Text	User manager

Start the project to test the application.

On the **UserManager** view, you can add users and edit or delete existing users.

Note:
You will not find any changes to users made at runtime in the user management editor in the IDE. To save these changes, the system creates a separate runtime file `<Projektname>.rtn` in the project directory.

Terminate the application.

Note:
The VisiWin 7 installation includes a user management sample project. To open the sample project, select **File > Samples > User Management > Open** in the IDE. Start the project and test the application at runtime. Terminate the application and take a closer look at the project in the IDE.

5.6 Recipes

HMI recipes define sets of process values that describe machine settings needed to manufacture a specific product of a product range. To write a recipe to the PLC means to switch to another product to be manufactured.

The *VisiWin Recipes system module* allows you to create *recipe classes* to which you can add variables that belong to a recipe. In the IDE, you can access the system module via the **Recipes** node in the project explorer. VisiWin includes a predefined view *RecipeView* that can be used to edit values of a recipe in the application.

In this tutorial, you will learn:

- how to create a recipe class,
- how to add the recipe edit view (RecipeView) to the project.

5.6.1 Create a recipe class

Recipe classes include variables, the values of which are saved at runtime and written to the PLC. Instead of using PLC variables, you are going to create two internal variables in this tutorial.

1. Create two new internal variables `RecipeVar1` and `RecipeVar2`.

   These variables will be used as simulated PLC variables.

2. Create a new recipe definition:
   a. In the project explorer, expand node **Recipes**.
   b. Double-click on the **Recipe Classes** node.

The table editor is now open in the workspace. As you have not defined any recipe classes yet, the table editor is empty.

c. Click on the table editor once and press **F8**.

d. In the **Property Pages** window, set the following parameters:

Parameter	Value
Name	TestRecipe
Activate change logging[3]	☑

A new recipe class named `TestRecipe` has been created.

3. Indicate the variables you want to include in the recipe:
   a. Right-click on the table editor and select **Select variables** from the context menu.

      The **Select Variable** dialog appears.

   b. Select internal variables *RecipeVar1* and *RecipeVar2*.

   c. Click **OK** to confirm.

   Variables *RecipeVar1* and *RecipeVar2* have been added as recipe elements to the `TestRecipe` class.

5.6.2 Add the RecipeView

The RecipeView template allows to manage and edit recipe files and to edit recipe values.

1. Add the RecipeView template to the project:
   a. In the project explorer, expand node **Design > Views**.
   b. Select node **MainRegion**.
   c. Right-click and select **Add new Element...** from the context menu.

      The **Add new Item** dialog appears.

   d. Select folder **Recipe** on the left-hand side of the dialog.

      The available templates are displayed on the right-hand side of the dialog. There is only one template in this case.

   e. Select project template **RecipeView**.

   f. Press **Enter** to confirm.

---

3. **Transfer behavior** tab

The **MainRegion** node now includes the RecipeView.xaml. Open the new view in the designer and have a look at it.

2. Bind a NavigationRadioButton to the RecipeView.xaml to enable users to navigate to the view:

   a. Open the **FooterView.xaml** in the designer.

   b. Select the NavigationRadioButton labeled with **MainView3**.

   c. Set the following control properties:

Property	Value
RegionName	MainRegion
ViewName	RecipeView
Text	Recipe view

3. Add NumericVarIn controls to the view that you can use to manipulate the values of *RecipeVar1* and *RecipeVar2*:
   a. Add two **NumericVarOut** controls to **MainView1.xaml**.
   b. Set the following properties for the first control:

Property	Value
LabelText	RecipeVar1
CanRecipeEdit	☐
VariableName	RecipeVar1

   c. Set the following properties for the second control:

Property	Value
LabelText	RecipeVar2
CanRecipeEdit	☐
VariableName	RecipeVar2

Start the project to test the application.

Your application should do the following:

1. PLC values are read into the recipe management and written from the recipe management to the PLC:
    a. Type any values into the **RecipeVar1** and **RecipeVar2** boxes on MainView1.
    b. Click on **Recipe view** to view the RecipeView.

    You cannot see the values on the **Recipe data** pane. These have to be read from the PLC first.

    c. Click on the **Load from PLC** button.

    Now you can see the values in the **Recipe data** list.

    d. Type other values on the **Recipe data** pane and click on **Send to PLC**.
    e. Click on the **MainView1** button to show MainView1.

    The new values are displayed in the **RecipeVar1** and **RecipeVar2** boxes on MainView1.

2. Recipes variable values changes are logged:
    a. In the RecipeView, click on the **Changes** button on the **File system** pane.

    A dialog appears where all changes to recipe variables are reported with a time stamp. You have enabled this function in the **Activate change logging** parameter of the recipe class.

3. Recipes variable values can be saved in recipe files:
    a. On the **Recipe Management** pane, type a name for the file you want to generate into the **Filename** box.
    b. Type a description for the file in the **Description** box, if desired.
    c. On the **File system** pane, click on **Save**.

    The values on the **Recipe data** pane have been saved in a recipe file.

    d. Save a second file under a different name and with different values.
4. Values saved in a recipe file can be loaded into the recipe management:
    a. On the **File system** pane, click on **Load**.
    b. Select the first recipe file.

    c. Click **OK** to confirm.

    Now you can see the values in the **Recipe data** list.

5. The recipe file values can be written from the recipe management to the PLC:
    a. Memorize the values displayed on the **Recipe data** pane.
    b. On the **File system** pane, click on **Send to PLC**.
    c. In the navigation menu, click on **MainView1**.

    The recipe file values are displayed in the input boxes. The have been transmitted to the “PLC”.

6. Recipe files can be deleted:
    a. On the **File system** pane, click on **Delete**.
    b. Select the recipe file you want to delete.
    c. Click **OK** to confirm.

    The recipe file has been deleted. If you click on **Load** now, the deleted recipe file is not on the list.

Terminate the application.

Note:
The VisiWin 7 installation includes a recipes sample project. To open the sample project, select **File > Samples > Recipes > Open** in the IDE. Start the project and test the application at runtime. Terminate the application and take a closer look at the project in the IDE.

5.7 Trends

The *VisiWin Trends system module* saves data for later representation and analysis.

A trend definition (*trend*) is created within the system module and contains a reference to the variable, the values of which will be saved at runtime. In the IDE, you can access the system module via the **Trends** node in the project explorer. To display trends as curves in a chart, VisiWin includes a *TrendChart* control and a predefined *TrendView*.

In this tutorial, you will learn:

- how to create trends,
- how to add a trend view (TrendView) to the project.

5.7.1 Create trends

A trend definition must belong to an *archive*. Archives define how trend data are stored.

You are going to use two variables from the **Ch1:Sample** channel. These variables constantly change their values so that you can see how trends are displayed.

1. Create a new archive:
    a. In the project explorer, expand node **Trends**.
    b. Select node **Archives**.
    c. Right-click and select **New** from the context menu.

You have created a new archive named *Archive1*. Within that archive, you can create trend definitions.

2. Create two trends within the archive:
    a. Click on the table editor once and press **F8** twice.
    b. In the **Name** parameter, type `Sawtooth` as a name for the first trend definition.
    c. Type `Sine` as a name for the second definition.

    You have created two trend definitions *Sawtooth* and *Sine*. Now you have to specify the trend variables.

3. Specify the variables for the two trends:
    a. In the first row, select the cell in the **Trend variable** column and click on the button.

        The **Select Variable** dialog appears.

    b. Expand node **Ch1 > AnimatedObjects > Sawtooth**.
    c. Select variable *Value*.

    d. Click **OK** to confirm.

    e. Select variable *Ch1.AnimatecObjects.Sine.Value* for the second definition.

You have created two trend definitions that you are going to add to a trend chart in the next step.

5.7.2 Add and edit the TrendView

The TrendView template includes a TrendChart control that displays curves in a chart.

1. Add the TrendView template to the project:
    a. In the project explorer, expand node **Design > Views**.
    b. Select node **MainRegion**.
    c. Right-click and select **Add new Element...** from the context menu.

        The **Add new Item** dialog appears.

    d. Expand folder **Charts** on the left-hand side of the dialog and select folder **TrendChart**.

        The available templates are displayed on the right-hand side of the dialog. There is only one template in this case.

    e. Select project template **TrendView**.

    f. Press **Enter** to confirm.

        The **MainRegion** node now includes the TrendView.xaml. If you open the TrendView in the designer, you will see no curves on the chart. The curves become visible if you bind trend variables to them.

In the **Document Outline** window, you can see the view contents. The view includes a TrendChart control that contains a TrendCurve2 element. You have to bind a trend definition to this element. To display a second trend, you have to add a second TrendCurve2 element and bind a trend definition to it, too.

2. Bind a trend definition to the first curve:
    a. In the **Document Outline** window, select the **TrendCurve2** element.
    b. In the **Properties** window, type **TrendName** into the search box.
    c. Click on the ... button next to the **TrendName** property.

    The **Select Trend** dialog appears.

    d. Select trend definition *Sawtooth*.

    e. Click **OK** to confirm.

    Now you can see a curve in the designer. Curves in the designer do not represent actual trend data.

3. Add a second curve and bind the second trend definition to it:
    a. In the **Document Outline** window, select the **TrendCurvesContainerDateTime** element.
    b. In the **Properties** window, type `Curves` into the search box.
    c. Click on the ... button next to the **Curves** property.

    The **Collection Editor** dialog appears. There is a curve on the **Items** list on the left-hand side of the dialog. You have already bound a trend variable to it.

    d. To add a second curve, click on **Add** below the **Items** list.
    e. Expand the **Daten** category on the right-hand side of the dialog.
    f. Click on the ... button next to the **TrendName** property.

    The **Select Trend** dialog appears.

    g. Select trend definition *Sine*.
    h. Click **OK** in both dialogs.

    A second curve appears in the designer.

4. Bind the TrendView.xaml to a NavigationRadioButton in the footer to enable users to navigate to the view:

    a. Open the **FooterView.xaml** in the designer.

    b. Select the NavigationRadioButton labeled with **MainView3**.

    c. Set the following control properties:

Property	Value
RegionName	MainRegion
ViewName	TrendView
Text	Trend view

Start the project to test the application.

Click on the **Trend view** button to see the trend chart on TrendView.xaml. The chart is recording two curves. The curve recording the values of the *Sine* trend definition is not always visible since the curve scale starts with 0 while the curve values range between 1 and -1.

In order to see all curve values, terminate the application and set the MaxValue and MinValue properties of the curve (second TrendCurve2 in the **Document Outline** window) to 1 and -1, respectively. After that, you will see a sine curve on the chart in the application.

Terminate the application.

Note:
The VisiWin 7 installation includes a trends sample project. To open the sample project, select **File > Samples > Trends > Open** in the IDE. Start the project and test the application at runtime. Terminate the application and take a closer look at the project in the IDE.