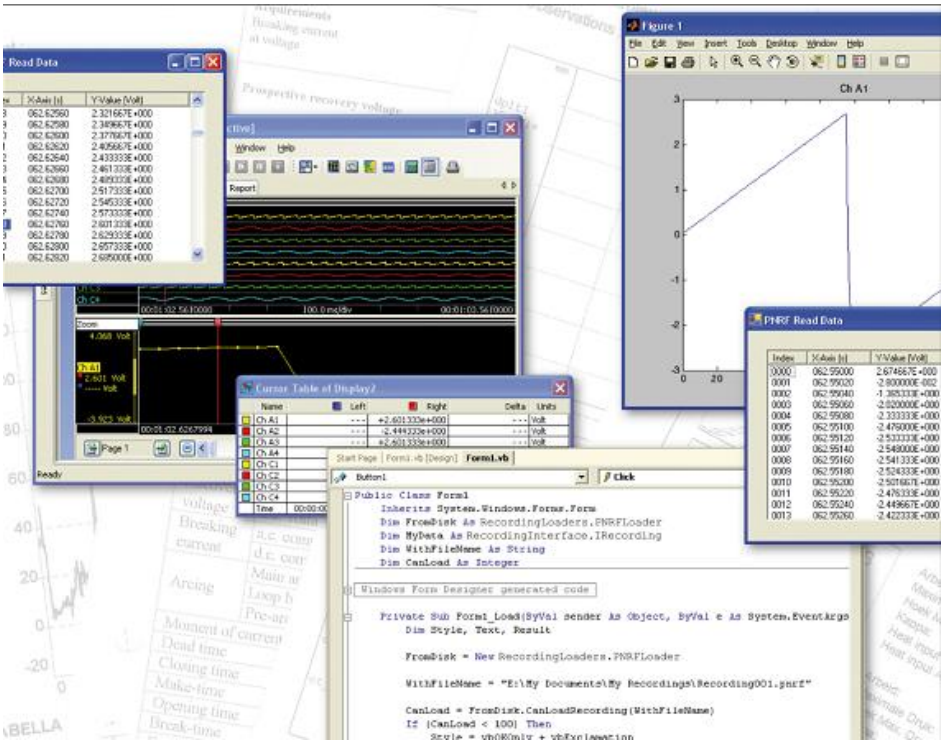
Genesis HIGH SPEED

Training

English



Perception PNRF Reader Training

HBM

Document version 1.0 – August 2011

For Perception 6.18 or higher

HBM GmbH
Im Tiefen See 45
64293 Darmstadt
Germany
Tel: +49 6151 80 30
Fax: +49 6151 8039100
Email: info@hbm.com
www.hbm.com/highspeed

LICENSE AGREEMENT AND WARRANTY

For more information about LICENSE AGREEMENT AND WARRANTY refer to:

www.hbm.com/terms

(Blank **Left** page only)

Table of Contents

LAB 1 – Creating your first PNRFReader program

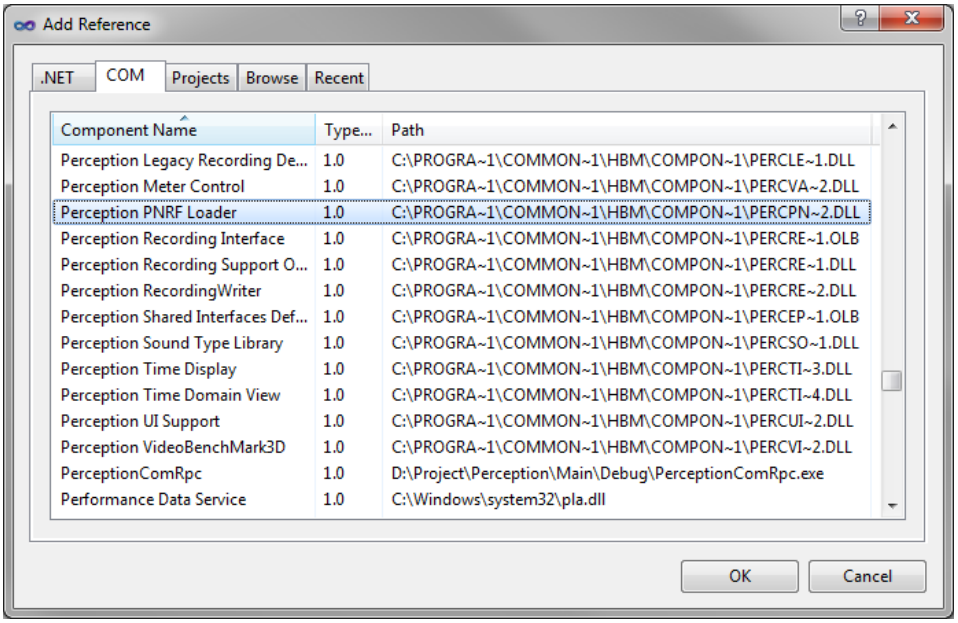
Start with *PnrfReader 1*

Purpose:

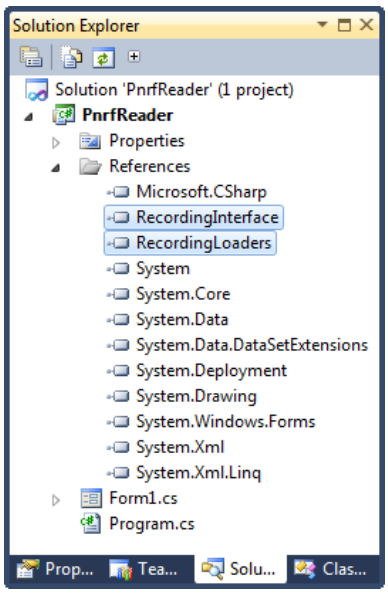
Creating pnrf reader program reading data samples and show the information or Data fields

Note: Before you start this Lab make sure you have a multi-sweep recording available, if not then create one with Perception and the Perception simulator

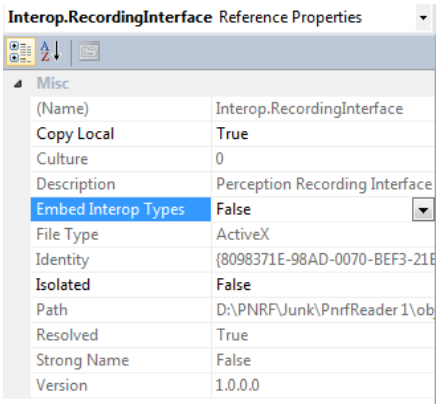
- Add a reference to the COM component called **Perception PNRF Loader**:



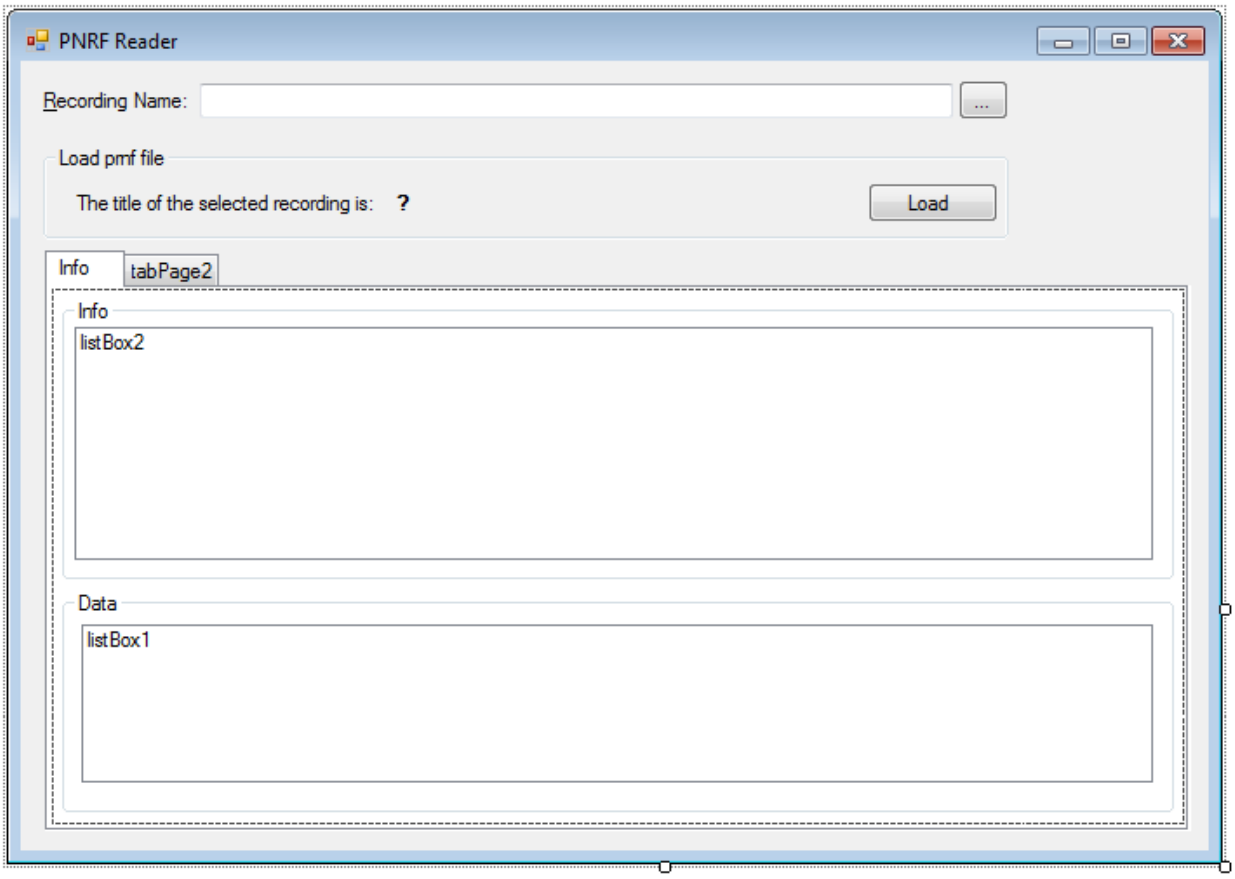
- The references are now added to the References tree:



The properties of the new references should be set as follow:



- Build the following user interface:




```
public partial class SheetControl : UserControl, ISheet, ICommon
```

- Add the following event code to the folder select (**...**) button click:

```
private void btnFolderSelect_Click(object sender, EventArgs e)
{
  OpenFileDialog Dialog = new OpenFileDialog();
  Dialog.Title = "Select Recording";
  Dialog.Filter = "Recording Files (pnrf)|*.pnrf";
  if (edtSourceFileName.Text != "")
  {
    if (File.Exists(edtSourceFileName.Text))
    {
```

```
      Dialog.FileName = edtSourceFileName.Text;
    }
  }

  if (Dialog.ShowDialog(this) == DialogResult.OK)
  {
    String myFileName = Dialog.FileName;
    if (myFileName.Length != 0)
    {
      if (File.Exists(myFileName))
      {
        try
        {
          edtSourceFileName.Text = myFileName;
        }
        catch
        {
          MessageBox.Show("Error reading File");
        }
      }
    }
  }
}
```

- Add the following event code to the **Load** button click:

```
private void btnLoad_Click(object sender, EventArgs e)
{
  if (!File.Exists(edtSourceFileName.Text))
  {
    MessageBox.Show("File " + edtSourceFileName.Text + " does not exist!");
    return;
  }
  ShowInfoData();
  ShowRecordingData();
}
```

- The **ShowInfoData**() function looks like:

```
private void ShowInfoData()
{
  listBox2.Items.Clear();
  PNRFLoader FromDisk = new PNRFLoader();

  string strRecordingFileName = edtSourceFileName.Text;
  IRecording myData = FromDisk.LoadRecording(strRecordingFileName);

  if (myData == null) return;
  if (myData.DataValues == null) return;

  listBox2.Items.Add(string.Format("D a t a V a l u e s ({0})",
          myData.DataValues.Count));
  for (int i = 1; i <= myData.DataValues.Count; i++)
  {
    IDataValue DataVal = myData.DataValues[i];
    listBox2.Items.Add(string.Format("Name: {0}, Value: {1} {2}, Type: {3}",
          DataVal.Name, DataVal.Value, DataVal.Units, DataVal.DataType));
  }

  listBox2.Items.Add(string.Empty);
  listBox2.Items.Add(string.Format("R e c o r d e r s ({0})",
                              myData.Recorders.Count));
  for (int i = 1; i <= myData.Recorders.Count; i++)
  {
    IDataRecorder myDataRecorder = myData.Recorders[i];
    string cPhysicalName = myDataRecorder.PhysicalName;
    string cGroup = myDataRecorder.Group.Name;

    int nTriggers = 0;
    if (myDataRecorder.Triggers != null)
```

```
    {
      nTriggers = myDataRecorder.Triggers.Count;
    }

    listBox2.Items.Add(string.Format("Recorder: {0} Group: {1} Triggers: {2}",
            cPhysicalName, cGroup, nTriggers));
  }
}
```

- The ShowRecordingData() looks like:

```
private void ShowRecordingData()
{
  PNRFLoader FromDisk = new PNRFLoader();

  string strRecordingFileName = edtSourceFileName.Text;
  IRecording myData = FromDisk.LoadRecording(strRecordingFileName);
  lblOutput.Text = myData.Title;
  listBox1.Items.Clear();

  if (myData.Channels.Count < 1)
  {
    MessageBox.Show("No Data");
    return;
  }
  IDataChannel myChannel = myData.Channels[1];
  listBox1.Items.Add(string.Format("Recording: {0}", myChannel.Recording.Title));
  listBox1.Items.Add(string.Format("Recorder: {0} Channel: {1}",
                  myChannel.Recorder.Name, myChannel.Name));
  IDataSrc myDataSrc =
              myChannel.get_DataSource(DataSourceSelect.DataSourceSelect_Mixed);

  double dStartTime = myDataSrc.Sweeps.StartTime;
  double dEndTime = myDataSrc.Sweeps.EndTime;

  listBox1.Items.Add(string.Format("Recording Start: {0}, End: {1}",
                  dStartTime, dEndTime));

  listBox1.Items.Add("");
  object mySegmentsData = null;
  myDataSrc.Data(dStartTime, dEndTime, out mySegmentsData);

  if (mySegmentsData == null)
  {
    MessageBox.Show("No Data");
    return;
  }

  IDataSegments mySegments = mySegmentsData as IDataSegments;
  if (mySegments == null)
  {
    MessageBox.Show("No Segments");
    return;
  }
  if (mySegments.Count < 1)
  {
    MessageBox.Show("No Segments");
    return;
  }

  IDataSegment mySegment = mySegments[1];

  object oRawData;
  mySegment.Waveform(DataSourceResultType.DataSourceResultType_Double64,
                          1, mySegment.NumberOfSamples, 1, out oRawData);

  if (oRawData == null)
  {
    MessageBox.Show("No Raw Data");
    return;
```

```
    }

    double[] aSamples = oRawData as double[];

    double X0 = mySegment.StartTime;
    double DeltaX = mySegment.SampleInterval;
    double X, Y;

    for (int i = 0; i < aSamples.Length; i++)
    {
      X = X0 + i * DeltaX;
      Y = aSamples[i];
      listBox1.Items.Add(string.Format("{0:000}:  X = {1}, Y = {2}", i, X, Y));

      // Do not show more then 500 samples
      if (i >= 500)
        break;
    }
  }
```

- The code is now ready; you can compile it and use the debugger to examine your code.

Result in *PnrfReader 2*

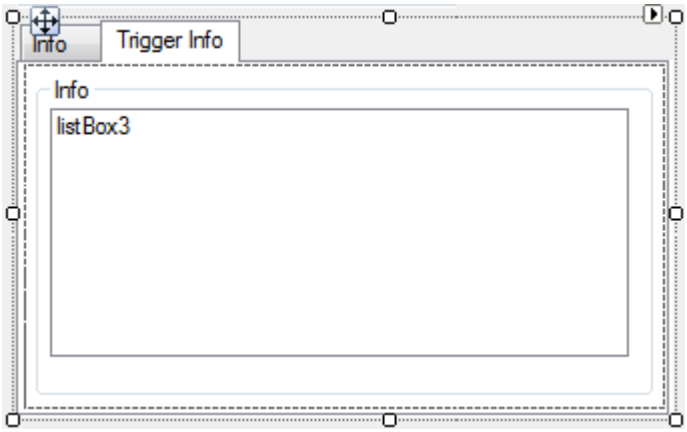
LAB 2 – Show basic Trigger information

Start with *PnrfReader 2*

Purpose:

Reading basic trigger information from pnrf file

- Add a list box to the 2nd page of the tab control



- Add the **ShowTriggerInfo()** function:

```
private void ShowTriggerInfo()
{
  PNRFLoader FromDisk = new PNRFLoader();
  string strRecordingFileName = edtSourceFileName.Text;
  IRecording myData = FromDisk.LoadRecording(strRecordingFileName);
  if (myData.Recorders.Count < 1) return;
  IDataRecorder myDataRecorder = myData.Recorders[1];
  if (myDataRecorder == null) return;
  if (myDataRecorder.Triggers == null) return;
  ITriggers myTriggers = myDataRecorder.Triggers;
  if (myTriggers.Count < 1) return;
  int nTriggerCount = myTriggers.Count;
  listBox3.Items.Clear();
  double dTime;
  TimeMarkType TriggerType;

  for (int i = 1; i <= nTriggerCount; i++)
  {
    ITimeMark myTimeMark = myTriggers[i];
    dTime = myTimeMark.Time;
    TriggerType = myTimeMark.MarkType;
    listBox3.Items.Add(string.Format("{0}: Trigger time: {1}, type: {2}",
            i, dTime, TriggerType));
  }
}
```

- Add a call to this function in the btnLoad_Click() function:

```
private void btnLoad_Click(object sender, EventArgs e)
{
  if (!File.Exists(edtSourceFileName.Text))
  {
    MessageBox.Show("File " + edtSourceFileName.Text + " does not exist!");
    return;
```

```
    }
    ShowInfoData();
    ShowRecordingData();
    ShowTriggerInfo();
}
```

- The code is now ready; you can compile it and use the debugger to examine your code.

Result in *PnrfReader 3*

LAB 3 – Show extra Trigger information

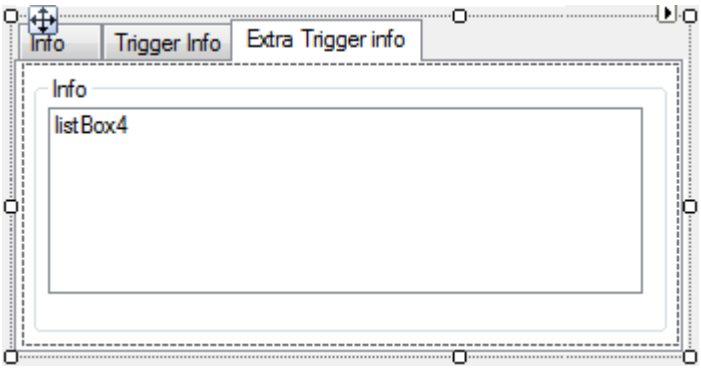
Start with *PnrfReader 3*

Purpose:

Reading extra trigger information from pnrf file.

This labs shows you how you can read the trigger time, the trigger start and end time and the trigger source.

- Add a list box to the 3th page of the tab control



- The **ShowExtraTriggerInfo()** looks like:

```
private void ShowExtraTriggerInfo()
{
  if (!File.Exists(edtSourceFileName.Text))
  {
    MessageBox.Show("File " + edtSourceFileName.Text + " does not exist!");
    return;
  }

  PNRFLoader FromDisk = new PNRFLoader();

  string strRecordingFileName = edtSourceFileName.Text;
  IRecording MyData = FromDisk.LoadRecording(strRecordingFileName);
  int nTriggerCount = 0; ;
  listBox4.Items.Clear();

  if (MyData.Recorders.Count < 1)
  {
    MessageBox.Show("No Recorders found");
    return;
  }
  if (MyData.Recorders[1] == null) return;
  if (MyData.Recorders[1].Triggers == null) return;
  nTriggerCount = MyData.Recorders[1].Triggers.Count;

  listBox4.Items.Add("Extended triggers");
  listBox4.Items.Add("");
  for (int Rec = 1; Rec <= MyData.Recorders.Count; Rec++)
  {
    for (int Sw = 1; Sw <= nTriggerCount; Sw++)
    {
      if (MyData.Recorders[Rec].Triggers[Sw].MarkType ==
                TimeMarkType.TimeMarkType_TriggerAnnotation)
      {
```

```
            listBox4.Items.Add(string.Format(
                "Recorder: '{0}' has Trigger: {1} time: {2} Type: {3}",
                MyData.Recorders[Rec].Name, Sw,
                MyData.Recorders[Rec].Triggers[Sw].Time,
                MyData.Recorders[Rec].Triggers[Sw].MarkType));
          }
        }
      }

      listBox4.Items.Add("");
      listBox4.Items.Add("Channel information");
      listBox4.Items.Add("");
      for (int Rec = 1; Rec <= MyData.Recorders.Count; Rec++)
      {
        if (MyData.Recorders[Rec].Channels != null)
        {
          listBox4.Items.Add(string.Format("Recorder: '{0}'",
                      MyData.Recorders[Rec].Name));
          if (MyData.Recorders[Rec].Channels.Count == 0)
          {
            listBox4.Items.Add("      No data");
          }
          else
          {
            for (int i = 1; i <= MyData.Recorders[Rec].Channels.Count; i++)
            {
              listBox4.Items.Add(string.Format("      Channel: '{0}'",
                            MyData.Recorders[Rec].Channels[i].Name));
              IDataSrc mySweepData =
                        MyData.Recorders[Rec].Channels[i].get_DataSource(
                        DataSourceSelect.DataSourceSelect_Sweeps);
              if (mySweepData != null)
              {
                if (mySweepData.Sweeps != null)
                {
                  // mySweepData.Sweeps.StartTime
                  listBox4.Items.Add(string.Format(
                              "         Sweep Start: {0} End: {1}",
                          mySweepData.Sweeps.StartTime, mySweepData.Sweeps.EndTime));
                  for (int j = 1; j <= mySweepData.Sweeps.Count; j++)
                  {
                    IDataSweep mySweep = mySweepData.Sweeps[j];
                    listBox4.Items.Add(string.Format(
                    "             Sweep[{0}]:  Start: {1}, End: {2}, Trigger Time: {3},
                        Trigger source {4}", j, mySweep.StartTime, mySweep.EndTime,
                        mySweep.TriggerTime, mySweep.TriggerSource));
                  }
                }
              }
            }
          }
        }
      }
    }
```

- Add a call to **ShowExtraTriggerInfo()** in the btnLoad_Click() function:
- The code is now ready; you can compile it and use the debugger to examine your code.

Result in *PnrfReader 4*

LAB 4 – Show mainframe settings

Start with *PnrfReader 4*

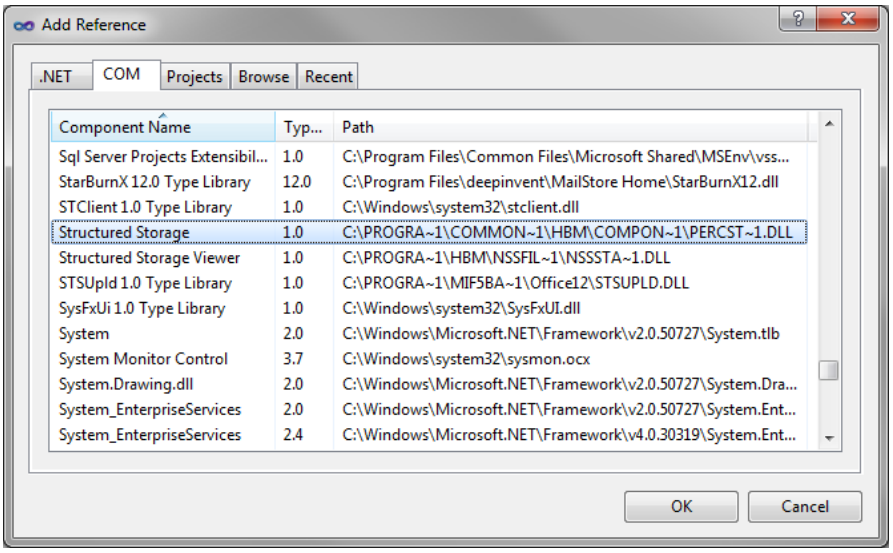
Purpose:

Reading the settings data from a pnrf file.

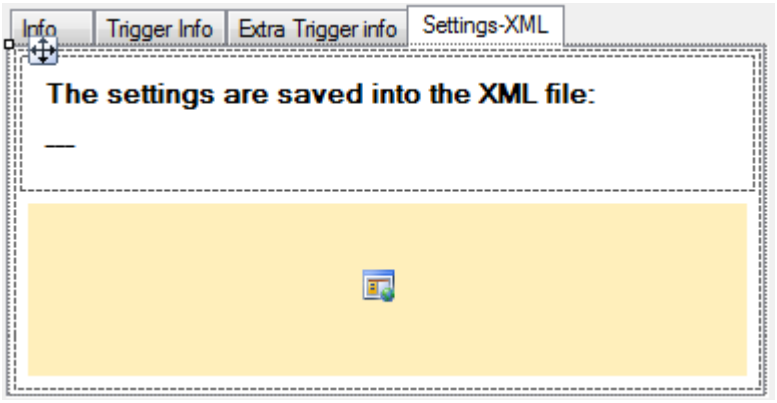
This labs shows you how you can read the settings data from a pnrf file. The settings are stored in a xml formatted stream.

Before you start coding use the **NSS Fileviewer** to have a look into the pnrf file you want to read.

Add a reference to the Structured Storage COM dll:



- Add a list box to the 4th page of the tab control
- Call this page Settings-XML
- Add a Panel and position it at the top of the page.
- Add two text labels to the panel (**lblXML1** and **lblXML2**)
- Add a webBrowser to the page, this component can be found in the common controls of the VS toolbox
- In the figure below you see the new UI, the yellow color is only drawn to show you where the webBrowser should be positioned, in your development environment it is white and therefore hard to find.



- Add the **ShowXMLSettingsData()** function:

```
private void ShowXMLSettingsData()
{
  PNRFLoader FromDisk = new PNRFLoader();

  string strRecordingFileName = edtSourceFileName.Text;
  IRecording myData = FromDisk.LoadRecording(strRecordingFileName);
  IExperiment myExperiment = myData as IExperiment;
  if (myExperiment == null)
  {
    return;
  }

  object pRootStorage;
  object pSystemStorage;
  myExperiment.GetStorages(out pRootStorage, out pSystemStorage);

  INSSStorage nssSystemStorage = pSystemStorage as INSSStorage;

  if (nssSystemStorage == null)
  {
    return;
  }

  INSSStorage SettingsStore = null;
  INSSStream SettingsStream = null;

  try
  {
    nssSystemStorage.OpenStorage("Settings",
        nssFlags.nssFlags_OpenStorageReadOnly | nssFlags.nssFlags_NoErrorReturn,
        out SettingsStore);
    if (SettingsStore == null)
    {
      return;
    }

    SettingsStore.OpenStream("Settings.xml", nssFlags.nssFlags_OpenStreamReadOnly |
                             nssFlags.nssFlags_NoErrorReturn, out SettingsStream);
    if (SettingsStream == null)
    {
      return;
    }

    IStream myIstream = SettingsStream as IStream;

    // Get the length of the stream
    System.Runtime.InteropServices.ComTypes.STATSTG pstatstg;
    myIstream.Stat(out pstatstg, 0);
    long nLen = pstatstg.cbSize;

    // Define a byte array to be used to store the stream
    byte[] myBytes = new byte[nLen];
```

```
      myIstream.Read(myBytes, myBytes.Length, IntPtr.Zero);

      // Convert the bytes array into a string, skip the the first two bytes
      string myXmlString = System.Text.Encoding.Unicode.GetString(myBytes, 2,
                  (int)nLen - 2);
      // show the xml string.
      LoadFromXMLString(myXmlString);
    }
    catch
    {
      return;
    }

    finally
    {
      ReleaseComObject(SettingsStream);
      ReleaseComObject(SettingsStore);
      ReleaseComObject(nssSystemStorage);
    }
  }
```

- Add the **LoadFromXMLString**() function:

```
public void LoadFromXMLString(string myXmlString)
{
  XmlDocument xmlDoc = new XmlDocument();
  xmlDoc.LoadXml(myXmlString);
  if (!xmlDoc.DocumentElement.HasChildNodes)
    return;
  XmlNode AcquisitionSystemNode = xmlDoc.SelectSingleNode("AcquisitionSystem");

  string myXMLFileName = edtSourceFileName.Text;
  myXMLFileName = Path.ChangeExtension(myXMLFileName, ".xml");
  lblXML2.Text = "'" + myXMLFileName + "'";
  xmlDoc.Save(myXMLFileName);
  webBrowser1.Url = new Uri(myXMLFileName);
}
```

- Add the **ReleaseComObject**() function:

```
public bool ReleaseComObject(object Obj)
{
  try
  {
    if (Obj == null || !Marshal.IsComObject(Obj))
      return false;

    Marshal.ReleaseComObject(Obj);
    return true;
  }
  catch
  {
    return false;
  }
}
```

- Add a call to **ShowXMLSettingsData()** in the btnLoad_Click() function:
- The code is now ready; you can compile it and use the debugger to examine your code.

Result in *PnrfReader 5*

LAB 5 – Reading any XML stream from a pnrf file

Start with *PnrfReader 6*

Purpose:

Reading the XML information from any stream in a pnrf file.

For this LAB you do not have to add code but you can start directly with the provided code of PnrfReader 6.

Go step by step through the code and try to understand what is happening.

Head Office
HBM
Im Tiefen See 45
64293 Darmstadt
Germany
Tel: +49 6151 8030
Email: info@hbm.com

France
HBM France SAS
46 rue du Champoreux
BP76
91542 Mennecy Cedex
Tél:+33 (0)1 69 90 63 70
Fax: +33 (0) 1 69 90 63 80
Email: info@fr.hbm.com

UK
HBM United Kingdom
1 Churchill Court, 58 Station Road
North Harrow, Middlesex, HA2 7SA
Tel: +44 (0) 208 515 6100
Email: info@uk.hbm.com

USA
HBM, Inc.
19 Bartlett Street
Marlborough, MA 01752, USA
Tel : +1 (800) 578-4260
Email: info@usa.hbm.com

PR China
HBM Sales Office
Room 2912, Jing Guang Centre
Beijing, China 100020
Tel: +86 10 6597 4006
Email: hbmchina@hbm.com.cn

measure and predict with confidence