# DEVELOPER WORKSHOP

ticketxpert.net™
web based help desk

**DOKUMENTATION**

**Workshop for creation of own PlugIns in TicketXPert.NET**


**INHALTSVERZEICHNIS**

# Inhaltsverzeichnis

# 1 Requirements

For the deployment of WorkflowManager-plug-ins for TicketXPert.NET you need a system with the following requirements:

- Visual Studio 2005 with the actual Service Pack,
- A secondary installation of the WorkflowManager-Service (a re-start of the service is necessary for the plug-in testing)

The following assemblies have to be available as reference for e new plug-in project:

- Isonet.Core.dll
- Isonet.SecurityManager.BusinessInterfaces.dll
- Isonet.WorkflowManager.BusinessInterfaces.dll
- Isonet.WorkflowManager.BusinessManagers.dll

You can copy these assemblies directly from the WorkflowManager directory.

Additionaly you need an extra Web-Reference to the TxpEnterprise WebServices. With the WebServices you have access to the ticket database. The WebServices are installed on the TxpEnterprise-Server and accessible by the "TxpEnterpriseWebServices".
A reference for the here available WebServices can be achievable by accessing the WebService by Internet Explorer.

# 2 XML definition of the plugin

The first step is the XML-definition of the plug-in. In this section we deal with the structure of the XML Code for the deployment of a plug-in.

## 2.1 BASIC XML STRUCTURE

The following example shows the XML code for the plugin "Status Change":

```
<?xml version="1.0" encoding="utf-8"?>
<Plugin namespace="Isonet.Workflow.Plugins.TXPE" name="StatusChange" objectgu-
id="1cb4ea153d304c95bc05c73a2f596fe1" category="ticketdatachange">
  <Version value="1">
    <Image namespace="Isonet.Workflow.Plugins.TXPE" object="StatusChange.Plugin" />
    <Status value="dev" />
    <SupportedLanguages>
      <Language value="de" />
      <Language value="en" />
      <Language value="fr" />
    </SupportedLanguages>
    <CustomProperties />
    <AllowedWorkflowTypes>
      <key value="*" />
    </AllowedWorkflowTypes>
    <Names>
      <Name language="de"><![CDATA[Statusänderung]]></Name>
      <Name language="en"><![CDATA[Status change]]></Name>
      <Name language="fr"><![CDATA[Modification du statut]]></Name>
    </Names>
    <Descriptions>
      <Description language="de"><![CDATA[Dieses Plugin ändert den Status des zugehörigen Ti-
ckets.]]></Description>
      <Description language="en"><![CDATA[This plugin changes the state of the assigned tick-
et.]]></Description>
      <Description language="fr"><![CDATA[Ce plugin modifie le statut du ticket correspon-
dant.]]></Description>
    </Descriptions>
    <Input>
      <Channels>
        <Channel name="InputID" targetworkflow="00000000000000000000000000000000" targetob-
ject="00000000000000000000000000000000" connectorguid="00000000000000000000000000000000"
forwardingmethod="move">
          <Names>
            <Name language="de">Eingang</Name>
            <Name language="en">Input</Name>
            <Name language="fr"><![CDATA[Entrée]]></Name>
          </Names>
        </Channel>
      </Channels>
      <Conditions />
    </Input>
    <Output>
      <Channels>
```

```
      <Channel name="OutputID" targetworkflow="00000000000000000000000000000000" targetob-
ject="00000000000000000000000000000000" connectorguid="00000000000000000000000000000000"
forwardingmethod="move">
        <Names>
          <Name language="de">Ausgang</Name>
          <Name language="en">Output</Name>
          <Name language="fr">Sortie</Name>
        </Names>
      </Channel>
    </Channels>
    <Conditions />
  </Output>
  <Actions>
    <Action order="1" name="State changeID" type="code" targetnames-
pace="Isonet.Workflow.Plugins.TXPE" targetobject="StatusChange.Plugin">
      <Names>
        <Name language="de">Statusänderung</Name>
        <Name language="en">State change</Name>
        <Name language="fr">Modification du statut</Name>
      </Names>
      <Instructions>
        <Language value="-">
          <Instructions>
            <Instruction name="TicketState" text="00000000000000000000000000000000">
              <Names>
                <Name language="de">Status</Name>
                <Name language="en">Status</Name>
                <Name language="fr">Statut</Name>
              </Names>
            </Instruction>
            <Instruction name="TicketStateFromAttachment" text="">
              <Names>
                <Name language="de">Status aus Attachment X</Name>
                <Name language="en">Status out of attachment X</Name>
                <Name language="fr"><![CDATA[Statuz de l''annexe x]]></Name>
              </Names>
            </Instruction>
          </Instructions>
        </Language>
      </Instructions>
    </Action>
  </Actions>
 </Version>
</Plugin>
```

You can see the basic structure:

- Basic definition in the <Plug-in>-node,
- Link to an icon (<Image>-node),
- Supported languages,
- Name and description in the supported languages,
- Inputs and outputs (each with 0 to n),
- Actions (normally just one),
- Instructions for the action.

## 2.2 CREATION OF A NEW PLUGIN

If you want to create a new plugin, you can use the XML code of an existing plugin. Afterwards you need to adapt the bold attributes for the new plugin:

- PlugIn NameSpace and Name, this have to add up to the filename of the xml file!
- PlugIn ObjectGuid, here you have to generate a new GUID.
- The XML file have to point to an icon, displaying in the WorkflowDesigner.
- Name of the action and a link to the plugin in the new assembly.

Additionaly you should adapt the following elements of the new plugin:

- Localized names and descriptions,
- Number of in- and outputs,
- The possible conditions for the in- and outputs,
- The necessary number of instructions for the new action.

## 2.3 CONDITIONS FOR THE IN- AND OUTPUTS

In some cases it is necessary to define conditions for in- and outputs. Here are some examples:

- Plug-in "Acceptance": If a user point-and-clicks "Yes", the item is forwarded to the outbox "Accepted"; if all users click "No", the item is forwarded to the outbox "Not Accepted".
- Decision for switches in the workflow, e.g. "Should the test case start?" → Yes/No

The XML element "Output" as an example for a task of a group:

```
<Output>
    <Channels>
     <Channel name="YesID" targetworkflow="00000000000000000000000000000000" targetobject="00000000000000000000000000000000" connectorguid="00000000000000000000000000000000" forwardingmethod="move">
       <Names>
         <Name language="de"><![CDATA[Ja]]></Name>
         <Name language="en"><![CDATA[Yes]]></Name>
         <Name language="fr"><![CDATA[Ja]]></Name>
       </Names>
     </Channel>
     <Channel name="NoID" targetworkflow="00000000000000000000000000000000" targetobject="00000000000000000000000000000000" connectorguid="00000000000000000000000000000000" forwardingmethod="move">
       <Names>
         <Name language="de"><![CDATA[Nein]]></Name>
         <Name language="en"><![CDATA[No]]></Name>
         <Name language="fr"><![CDATA[Nein]]></Name>
       </Names>
     </Channel>
    </Channels>
    <Conditions>
```

```
    <Condition channel="NoID">
     <ResultGroup>
      <Result type="action" id="TaskID" value="false" number="100%">
       <Names>
        <Name language="de">Aufgabe</Name>
        <Name language="en">Task</Name>
        <Name language="fr">Aufgabe</Name>
       </Names>
      </Result>
     </ResultGroup>
    </Condition>
    <Condition channel="YesID">
     <ResultGroup>
      <Result type="action" id="TaskID" value="true" number="1">
       <Names>
        <Name language="de">Aufgabe</Name>
        <Name language="en">Task</Name>
        <Name language="fr">Aufgabe</Name>
       </Names>
      </Result>
     </ResultGroup>
    </Condition>
   </Conditions>
  </Output>
```

You can see, that a condition is defined for every output. Every condition has a reference to the name of the channels (“YesId“ and “NoId“).
The conditions themselves should depend on the result of the workflow activity so that the <Result>-node is specified with the attributes *type=”action”* and *id=”TaskID”*. Beneath you can see the <Action>-node. The attributes *value=”true“* and *value=”false“* refer to the outcome of the choice in the <Instruction>-node during the workflow action (see below). The attribute “number” contains the number of results of the denoted values (here “true” and “false”). You can specify absolute values (e.g. 1 or 6) or relative values (e.g. 25% or 100%)

The next example shows the <Action>-node. All not required elements (like localized texts) are hidden:

```
<Action order="1" startupaction="false" enabled="true" name="TaskID" type="todo" targetnamespace="Isonet.Workflow.Plugins.TXPE" targetobject="CommonTaskYesNoGroup.Plugin">
 <Names>
  <Name language="de">Task</Name>
  <Name language="en">Aufgabe</Name>
 </Names>
 <Instructions>
  <LanguageIndependent />
  <Localized>
   <Instructions>
    <Instruction value="true" name="option">
     <DesignerNames>
      <Name language="de">Option 'Ja'</Name>
      <Name language="en">Option 'Yes'</Name>
     </DesignerNames>
     <Texts>
      <Text language="de">Ja</Text>
      <Text language="en">Yes</Text>
```

```
        </Texts>
       </Instruction>
       <Instruction value="false" name="option">
        <DesignerNames>
         <Name language="de">Option 'Nein'</Name>
         <Name language="en">Option 'No'</Name>
        </DesignerNames>
        <Texts>
         <Text language="de">Nein</Text>
         <Text language="en">No</Text>
        </Texts>
       </Instruction>
     </Instructions>
   </Localized>
  </Instructions>
</Action>
```

The following example shows the condition for an input-channel. Such conditions are deployed in plug-ins which should synchronize parallel activities or merge WorkflowItems.

```
<Conditions>
 <Condition channel="Input1ID">
  <ResultGroup>
   <Result type="channel" id="Input1ID" value="true" number="1" />
   <Result type="channel" id="Input2ID" value="true" number="1" />
  </ResultGroup>
 </Condition>
 <Condition channel="Input2ID">
  <ResultGroup>
   <Result type="channel" id="Input1ID" value="true" number="1" />
   <Result type="channel" id="Input2ID" value="true" number="1" />
  </ResultGroup>
 </Condition>
</Conditions>
```

A further example:

```
<Conditions>
 <Condition channel="">
  <ResultGroup>
   <Result type="channel" id="[any]" value="true" number="100%" />
  </ResultGroup>
 </Condition>
</Conditions>
```

If the conditions are not defined in the input-channels, the WorkflowItems are directly passed to the plug-in where the action is executed. Only with input conditions a joint of multiple WorkflowItems can be effected.

# 3 Plugin Assembly

For the assembly creation of a plug-in you need to create a new .NET 2.0 project (C# Class Library) or a new namespace in an existing project of Visual Studio 2005.

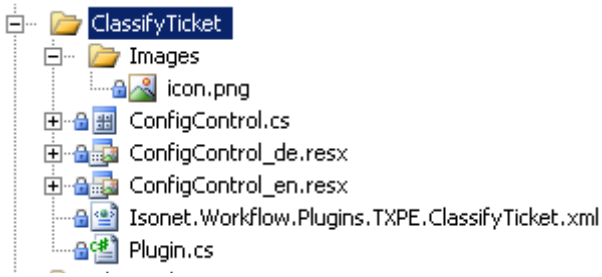
Add the references to the following assemblies:

- Isonet.WorkflowManager.BusinessManagers.dll,
- Isonet.Core.dll,
- Isonet.SecurityManager.BusinessInterfaces.dll,
- Isonet.WorkflowManager.BusinessInterfaces.dll.

Components of the new plug-in in the .NET-project:

- Plug-in.cs,
- Usercontrol for canfiguration,
- Icon.png,

The plug-in.xml (for a better organization of the files).




## 3.1 CODE FILE “PLUGIN.CS“

The class “Plug-in.cs” has to derive from the base-class “WorkflowPlug-inBase” in the “Isonet.WorkflowManager.BusinessInterfaces”.

The constructor has to be nameless the overwriting of the following methods are mandatory:

- Public override void Execute(IWorkflowItem item, IAction action),
- Public override DataSet GetList(IInstruction instruction, Guid clientObjectGuid).

The execute-method is accessed thru the plug-in-runtime. At the WorkflowItems-instance you can access directly the object running in the workflow. At the instance-type “IAction“ you can access the configuration of the action.

## Example for the execute-method of the plug-in "Status Change":

```
public override void Execute(IWorkflowItem item, IAction action)
{
	// Objekt for the creation of logentries in the isonet logfile
	Logger logger = new Logger();

	IInstructionCollection instructions = action.InstructionsLanguageIndependent;

	Guid ticketStateObjectGuid = Guid.Empty;
	string ticketStateObjectGuidFromAttachment = string.Empty;

	IInstruction instruction = null;
	for (int i = 0; i < instructions.Count; i++)
	{
		instruction = instructions[i];

		if (instruction.Name == "TicketState" && instruction.LanguageIndependentText.Length >
0)
		{
			ticketStateObjectGuid = new Guid(instruction.LanguageIndependentText);
		}

		if (instruction.Name == "TicketStateFromAttachment" && instruc-
tion.LanguageIndependentText.Length > 0)
		{
			ticketStateObjectGuidFromAttachment = instruction.LanguageIndependentText;
		}
	}

	// where required include from attachment
	if (ticketStateObjectGuidFromAttachment.Length > 0)
	{
		if (item.ContainsDataField(ticketStateObjectGuidFromAttachment) &&
item.GetDataField(ticketStateObjectGuidFromAttachment).Length > 0)
		{
			try
			{
				Guid g = new
Guid(item.GetDataField(ticketStateObjectGuidFromAttachment));
				if (g != Guid.Empty)
				{
					ticketStateObjectGuid = g;
				}
			}
			catch (Exception) { }
			item.SetDataField(ticketStateObjectGuidFromAttachment, string.Empty);
		}
	}

	if (ticketStateObjectGuid == Guid.Empty)
	{
		throw new ApplicationException("No ticketStateObjectGuid specified.");
	}

	TicketStoreService.TicketStoreService tss = new TicketStoreService.TicketStoreService();
	tss.SetStatusNoNotificationByDataId(item.Parent.ClientObjectGuid.ToString("n"),
int.Parse(item.IdUniqueForType), ticketStateObjectGuid.ToString("n"), ObjectTypeGu-
id.Workflowmanager.ToString("n"));
}
```

This plugin changes the status of a ticket to the static one denoted in the instruction "TicketState", or to a dynamic one in the data field "SavedTicketState". The change occurs thru a webservice.
For comparison, here the associated XML-definition of this action:

```
<Action order="1" name="State changeID" type="code" targetnames-
pace="Isonet.Workflow.Plugins.TXPE" targetobject="StatusChange.Plugin">
 <Names>
  <Name language="de">Statusänderung</Name>
  <Name language="en">State change</Name>
 </Names>
 <Instructions>
  <Language value="-">
   <Instructions>
    <Instruction name="TicketState" text="00000000000000000000000000000000">
     <Names>
      <Name language="de">Status</Name>
      <Name language="en">Status</Name>
     </Names>
    </Instruction>
    <Instruction name="TicketStateFromAttachment" text="SavedTicketStatus">
     <Names>
      <Name language="de">Status aus Attachment X</Name>
      <Name language="en">Status out of attachment X</Name>
     </Names>
    </Instruction>
   </Instructions>
  </Language>
 </Instructions>
</Action>
```

As a reminder: The "TargetNameSpace" and the "TargetObject" in the <Action>-node have to fit the name-space and the class-name of the new plug-in.

The method "GetList" is accessed during the configuration-time of the plug-in. Depending on the changed instructions the equivalent preset date is returned. In a particular UserInterface you can revert to this method (see section 4).

An example for the GetList-method in the plug-in "Status Change": In case the instruction "TicketState" is changed, a list of all predefined ticket states of TicketXPert.NET is outputted.

```
public override DataSet GetList(IInstruction instruction, Guid clientObjectGuid)
{
        DataTable dt;

        switch (instruction.Name)
        {
                case "TicketState":
                        dt = getTicketStateList(clientObjectGuid);
                        break;
                default:
                        dt = getEmptyDataTable();
                        break;
        }

        DataSet ds = new DataSet();
```

```
            ds.Tables.Add(dt);
            return ds;
}

private DataTable getTicketStateList(Guid clientObjectGuid)
{
            DataTable dt = getEmptyDataTable();

            TicketStateService.TicketStateService tss = new TicketStateService.TicketStateService();
            DataSet ticketStates = tss.GetTicketStates(clientObjectGuid, this.language);
            DataTable ticketStateTable = ticketStates.Tables[0];
            DataView ticketStateView = new DataView(ticketStateTable);
            ticketStateView.Sort = "Name ASC";

            foreach (DataRowView ticketState in ticketStateView)
            {
                        dt.Rows.Add(new object[] { ticketState["ObjectGuid"].ToString(), ticketS-
tate["Name"].ToString() });
            }

            return dt;
}
```

Please keep in mind: The list of the available states is provided by web service.

## 3.2 HANDLING OF THE WORKFLOWITEM

The WorkflowItem is given the method “Execute“ as a parameter to take with it. The WorkflowItem does not fit a ticket, in fact it is an autonomous object, carried thru the workflow by the Workflowmanager and containing sufficient information to define the represented ticket. Independent of the ticket, it can have its own creator, owner and affected user. Use the following properties:

- AffectedUserObjectGuid: Guid of the affected user,
- ConnectorObjectGuid: Contains the Guid of the workflow-channel (arrow in the workflow designer), if the item is located currently between two plug-ins. If it is located inside a plug-in (the most time), the field is empty.
- CreatorUserObjectGuid: Contains the Guid of the creator.
- DataId: Primary database-key of the item
- IdDescriptive: Descriptive form of the referenced object. On a ticket it is the ticket number (e.g. IN_00234).
- IdUniqueForType: Unique identifier of the referenced object.
- IsErroneous: This is “True” if the item “hangs” in the workflow (displayed red in the designer).
- LastException: Contains the last exception if IsErroneous=True.
- ObjectType: Object-type of the referenced object. If it is a ticket, the field contains the value “ticket”.
- OwnerGroupObjectGuid: The current owner group.
- OwnerUserObjectGuid: Contains the Guid of the actual owner. The field is empty if the ticket is assigned only to a group.

- Parent: Hands out the instance of the ItemGroup on which the item belongs. You can access the ClientObjectGuid by the ItemGroup, this action is frequently required by accessing a web service.
- PendingActionResultSets: Allows the access to the workflow activities of the users.
- Plug-inObjectGuid: Guid of the plug-in in which the item is located. Here it is about the InstalledObjectGuid.
- PreviousConnectorObjectGuid:
- PreviousPlug-inObjectGuid:
- PreviousWorkflowObjectGuid: These three properties return the last plug-in, the item passed through.
- SuspendedUntil: If IsSuspended=True, SuspendedUntil delivers the date and the time until the item is in wait mode.
- WorkflowObjectGuid: Guid of the current workflow containing the item.

Overview about capital available methods:

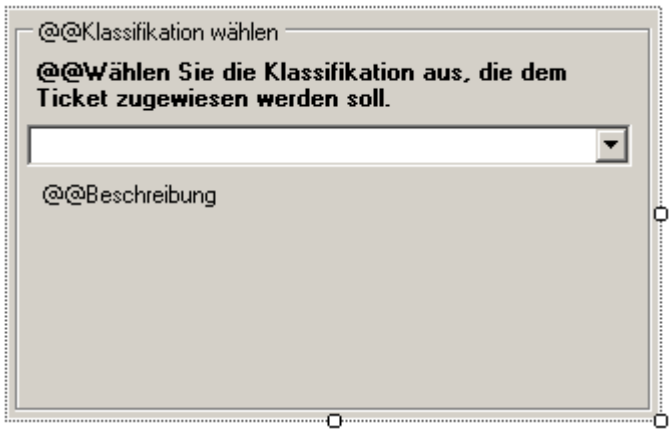
- ContainsDataField: Send back “True” if the denoted data field exists.
- GetAllBinaryAttachments: Send back all binary attachments.
- GetAllTextAttachments: Send back all available text attachments.
- GetDataField: Send back the content of the denoted data field.
- Resume: Fetch the item from the wait status back.
- SetDataField: Sets a data field to the specific value.
- SetIds: Changes the Id of the WorkflowItem.
- SetImmediateForwardLocation: The WorkflowManager relocates the item to the specific position soonest.
- SetUsers: Sets the user specified in the WorkflowItem.
- Suspend: Set the status of the item to waitmode.

### 3.3 ICON.PNG

The item, displayed in the WorkflowDesigner, have to be located in the subdirectory “Images” and have to be named “Icon.png“. Please note that we denote „Build Action“ (you can find it in the properties of the file „Icon.png“) to „Embedded Resource“.
The icon should have a transparent background.

# 4 The UserInterface for the Plugin Configuration

For a better ability of configuration of the plug-in you can add an own user-control. Add a file of the type “UserControl” and choose an adequate name. Design the UserControl exactly, as you would design other Windows-Forms-Applications:




Example from the plugin “Classify Ticket“

You need not add Controls for “Ok” or “Cancel” - they will added automatically by the WorkflowDesigner

Implement the interface “ICustomConfigControl” to your user-control in the name-space “Isonet.WorkflowManager.BusinessInterfaces.WorkflowPlug-in“. You have also to implement the following interface members:

- public void Init(Guid clientObjectGuid, IPlugin plug-in, string uiLanguage)
- public void Save()

The “Init”-method is executed before the UserControl is displayed in the workflow-designer. The request of the “Save“-method occurs after the click on the “OK“-button.

## 4.1 LOCALIZED TEXTS

If it is necessary to display localized texts, proceed in the following way:

- Choose the control which should own localized texts (e.g. “MyLabel“).
- Add the following code-line to the init-method:
  MyLabel.Text = CustomConfigControlBase.GetText(this.GetType(), "**MyLabel.Text**");
- Create a recource file for every language and add it to your project. The recource file should be named in the following way:

[NameOfYourControl]_[Language].resx
Example: ConfigControl_de.resx or ConfigControl_en.resx

- Add for everey localized text an entry to your recource file. The name has to match the name used in the code (see above, bold).

| | Name | Value |
|---|---|---|
| ▶ | MyLabel.Text | Anzuzeigender Deutscher Text |

## 4.2 EMBEDDING OF THE USERCONTROL

To embed the UserControl to your Plug-in, you have to overwrite two more properties:

- public override bool HasCustomConfigControl
  Just specify here: "True".
- public override Type CustomConfigControlType
  Specify the type of your UserControl.

Example:

```
public override bool HasCustomConfigControl
{
        get
        {
                return true;
        }
}

public override Type CustomConfigControlType
{
        get
        {
                return typeof(ConfigControl);
        }
}
```

## 4.3 CODE EXAMPLE

```
public void Init(Guid clientObjectGuid, IPlugin plugin, string uiLanguage)
{
        this.plugin = plugin;
        IInstruction instruction = plugin.Actions["ClassifyTicket"].InstructionsLanguageIndependent[0];
        this.initControls(clientObjectGuid, instruction, uiLanguage);

        this.groupBox1.Text = CustomConfigControlBase.GetText(this.GetType(), "groupBox1.Text");
        this.lblClassification.Text = CustomConfigControlBase.GetText(this.GetType(), "lblClassification.Text");
}

public void Save()
{
        IInstruction instruction = this.plugin.Actions["ClassifyTicket"].InstructionsLanguageIndependent[0];
        instruction.LanguageIndependentText = ((ClassificationListI-
tem)this.drpClassifications.SelectedItem).DataId.ToString();
}

private void initControls(Guid clientObjectGuid, IInstruction instruction, string uiLanguage)
{
        int classificationDataId = 0;
```

```
        if (instruction.LanguageIndependentText != null && instruction.LanguageIndependentText.Length > 0)
        {
                classificationDataId = int.Parse(instruction.LanguageIndependentText);
        }

        this.drpClassifications.Items.Clear();

        string nullName =  CustomConfigControlBase.GetText(this.GetType(), "nullName");
        string nullDescription =  CustomConfigControlBase.GetText(this.GetType(), "nullDescription");
        ClassificationListItem nullItem = new ClassificationListItem(0, nullName, nullDescription);
        this.drpClassifications.Items.Add(nullItem);
        this.drpClassifications.SelectedItem = nullItem;

        IPluginRepository pluginRepository = (IPluginReposito-
ry)Isonet.Core.Helpers.Remoting.ObjectProvider.GetObject(typeof(IPluginRepository));
        DataTable classificationsTable = pluginRepository.GetParameterListOfAction(instruction, uiLanguage,
clientObjectGuid).Tables[0];

        foreach (DataRow classification in classificationsTable.Rows)
        {
                int dataId = int.Parse(classification["Id"].ToString());
                ClassificationListItem item = new ClassificationListItem(
                        dataId,
                        classification["Name"].ToString(),
                        classification["Description"].ToString());

                this.drpClassifications.Items.Add(item);
                if (dataId == classificationDataId)
                {
                        this.drpClassifications.SelectedItem = item;
                }
        }

        this.drpClassifications_SelectedIndexChanged(this.drpClassifications, new EventArgs());
}

private struct ClassificationListItem
{
        public int DataId;
        public string Name;
        public string Description;

        public ClassificationListItem(int dataId, string name, string description)
        {
                this.DataId = dataId;
                this.Name = name;
                this.Description = description;
        }

        public override string ToString()
        {
                return Name;
        }
}

private void drpClassifications_SelectedIndexChanged(object sender, EventArgs e)
{
        ClassificationListItem item = (ClassificationListItem)((ComboBox)sender).SelectedItem;
        this.lblDescription.Text = item.Description;
}
```

# 5 Testing of the plugin

To test the plug-in the following conditions have to be fulfilled:

- XML definition of the plugin in the directory “WorkflowManager\PlugIns“,
- Assembly (DLL) of the plugin in the directory “WorkflowManager\PlugIns“,
- Restart of the WorkflowManager-service in the canagement console,
- Restart of the WorkflowDesigner.

The new or enhanced plugin appears in the WorkflowDesigner.

Warning: If you have references to WebServices inc luded in your plug-in (so-called WebReferences), the URL should be configurable. You can do this in the plug-in-configuration or by embedding the WebService as “dynamic” (in the properties). Visual Studio 2005 generates automatically a file “app.config” in the project and adds the proper ApplicationSettings.
If you want to test your plug-in, you have to add an entry of the Application-Settings to the configuration file of the WorkflowDesigner “Iso-net.WorkflowManager.exe.con-fig”
But we suggest to realize the URL for your own WebService as a configurable parameter.

# 6 Statistics and change management

## 6.1 STATISTICS

| | |
|---|---|
| From | Alexander Schmidt |
| Date | 12.01.2007 |
| Notice to | zuständiges Betriebsteam der isonet |
| Dok-ID | TXP-2100981 |
| Version | 1.1 |
| Status | Released |
| Replaced Version | - |
| Release date | 05.03.2007 |
| Valid from | Immediately |
| Valid until | Cancellation |
| Dokument title | Workflow PlugIn Development.doc |

## 6.2 CHANGE CONTROL

| Version | Date | Executed by | Comments/Type of change |
|---|---|---|---|
| 1.0 | 12.01.2007 | Alexander Schmidt | New |
| 1.1 | 05.03.2007 | Alexander Schmidt | Changed |
| 1.1 | 10.10.2008 | Adrian Karlen | Checked and overwrited |

## 6.3 INSPECTIONS

| Version | Date | Inspected by | Comments |
|---|---|---|---|
| 1.0 | 13.01.2007 | Thomas Luck | Geprüft |
| 1.1 | 05.03.2007 | Thomas Luck | Geprüft |

## 6.4 RELEASE

| Version | Date | Released by | Comments |
|---|---|---|---|
| 1.1 | 05.03.2007 | Thomas Luck | Released |

# 7 Non Disclosure Agreement

All documents that you receive concerning this project have to be handled explicitly and completely confidentially. It is not allowed to publish or reuse these documents as a whole or in excerpts.