
Integrate a new Tool into pure::variants by using VEL

Table of Contents

1. Overview	1
2. About this tutorial	1
3. Setup of the programming environment	1
3.1. Install needed Software	1
3.2. Preparation	1
4. Level 1 Integration	6
4.1. Create pure::variants Restriction Action	6
4.2. Write VEL Description	10
4.3. Process VEL Configuration	14
4.4. Create pure::variants Transformation Module	19
5. Level 2 Integration	24
5.1. Embed a pure::variants Integration User Interface	24
5.2. pure::variants Restriction Action	26
5.3. Preview	27

1. Overview

This tutorial shows how to extend **pure::variants** with a new tool connector using the **Variability Exchange Language**. It uses the open source mind mapping tool FreeMind to demonstrate the whole process. In the following section, we'll show how to add variability information into the mind map by using the standard editing components provided by **pure::variants**. We'll also connect the tool into the transformation framework of **pure::variants**. Finally, we'll use advanced integration components to support variant preview in FreeMind.

2. About this tutorial

The reader of this tutorial is expected to have basic knowledge about and experiences with **pure::variants** as well as how connectors work in Java programming. This tutorial is available in the Eclipse help section and in printable PDF format [here](#).

3. Setup of the programming environment

3.1. Install needed Software

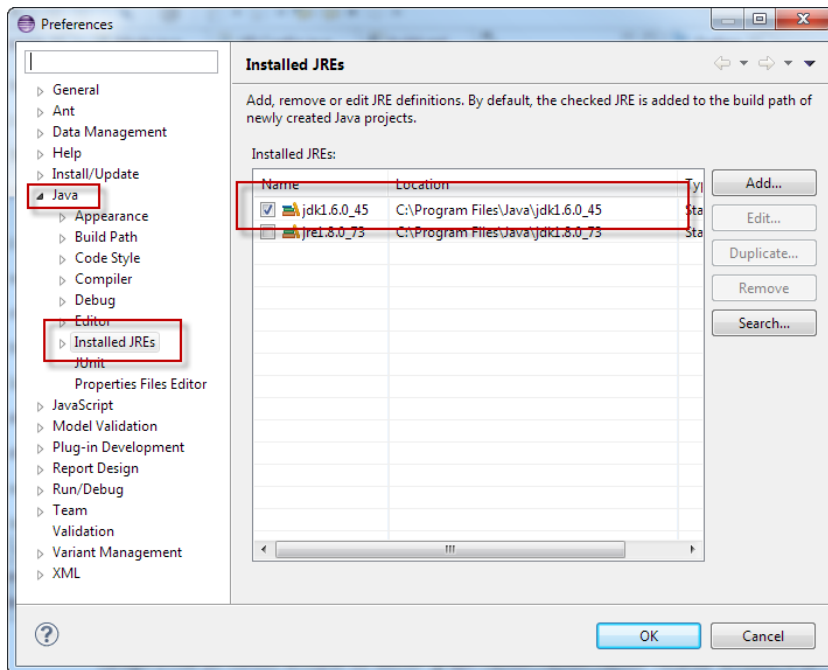
In order to run **pure::variants** and FreeMind you need to have Java installed on your machine. For extending FreeMind exactly Java 1.6 is needed. Please install the latest available version of [Java 1.6](#) on your operating system.

You'll also need **pure::variants** installed on your machine. This tutorial requires **pure::variants** 4.0.3 or later. After installing **pure::variants**, please make sure that the SDK and the VEL feature are installed.

3.2. Preparation

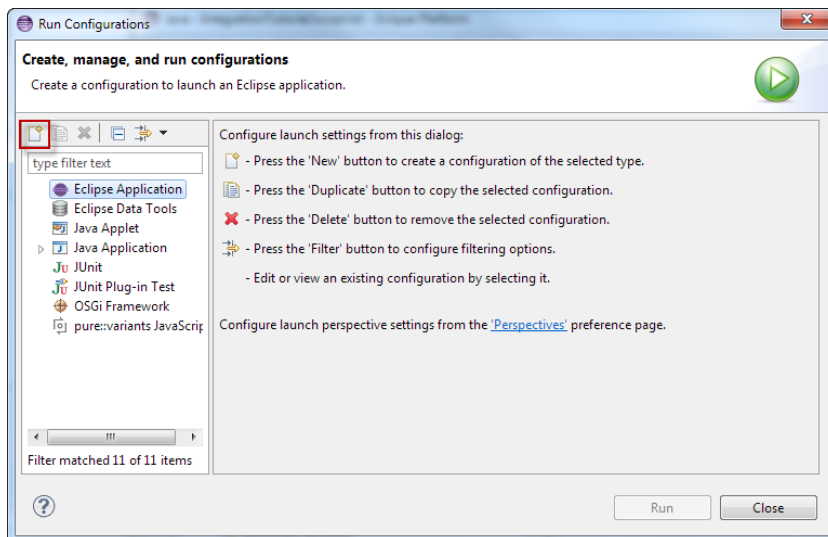
As the first step, select Java 1.6 from the installed JREs list.

Figure 1. Set JRE to Java 1.6



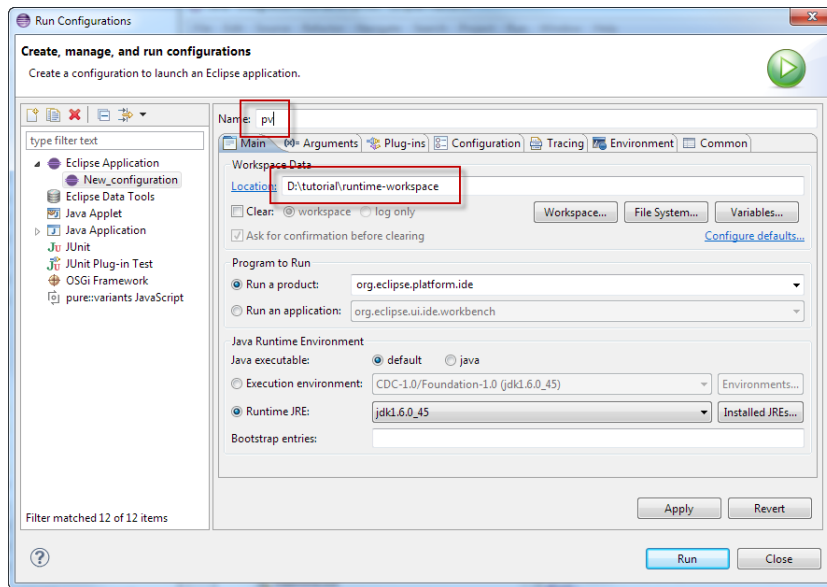
Setup a pure::variants runtime. We'll need this later to develop the transformation module. We'll also use the corresponding workspace for the project data. Switch to the *Java* perspective and create the runtime configuration with *Run -> Run Configurations...* Select *Eclipse Application* and press the *New* launch configuration button.

Figure 2. Create runtime configuration



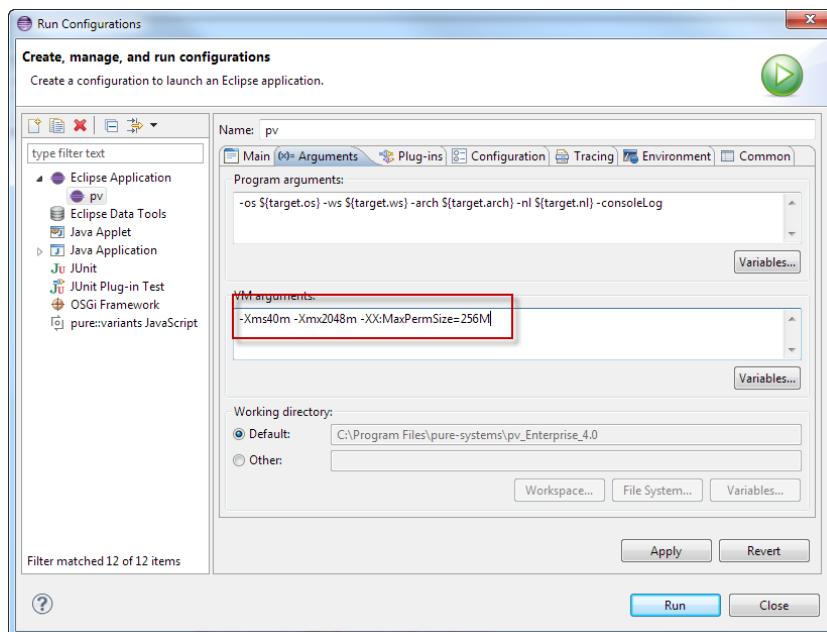
Enter a name and the workspace location.

Figure 3. Create runtime configuration



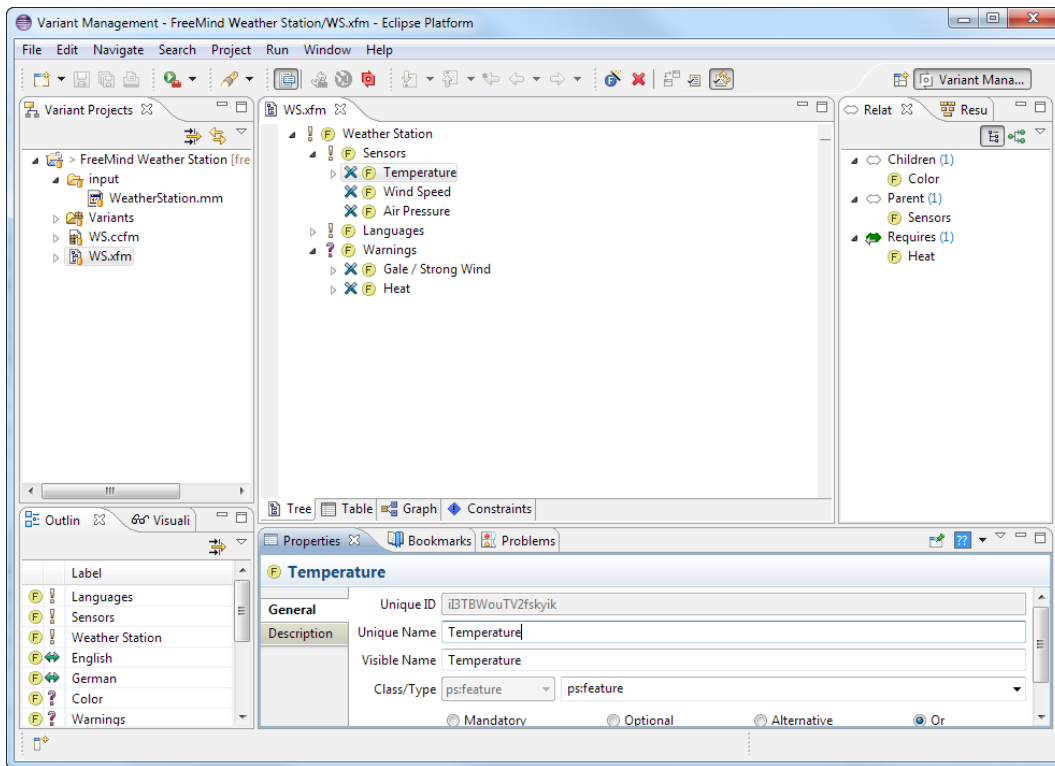
Add `-XX:MaxPermSize=256M` to the VM Arguments field.

Figure 4. Create runtime configuration



Start the runtime by pressing the *Run* button. A new Eclipse workbench with empty workspace starts. Please import the [FreeMind Weather Station project](#) into the workspace. The project contains the weather station models and a simple FreeMind mind map.

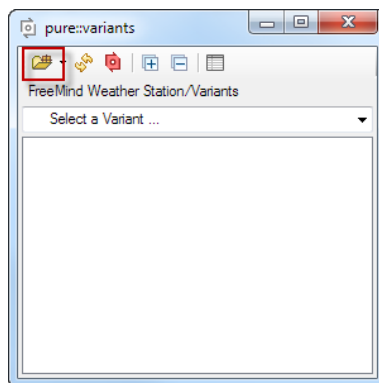
Figure 5. pure::variants runtime with example project



Close the eclipse runtime. You'll be back in your development Eclipse.

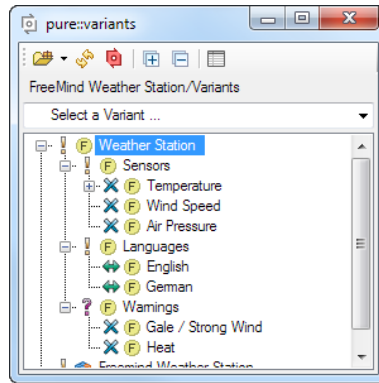
In the first integration level, we'll use the pure::variants Desktop Hub for restriction editing. Open the start menu and select the *Desktop Hub* item in the pure::variants start menu folder. In the Windows task bar a pure::variants icon will appear. Double-click that icon to open the Desktop Hub. Please load the project from the runtime workspace into the Desktop Hub.

Figure 6. Load project into pure::variants Desktop Hub



In the file dialog, navigate to the project in the runtime workspace. Select the `configspace.xml` file in the `Variants` folder. The Desktop Hub will now load the models and show the feature tree.

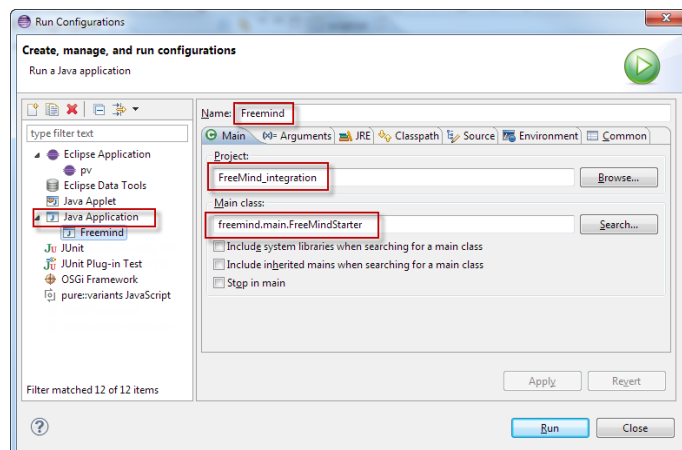
Figure 7. pure::variants Desktop Hub with project models



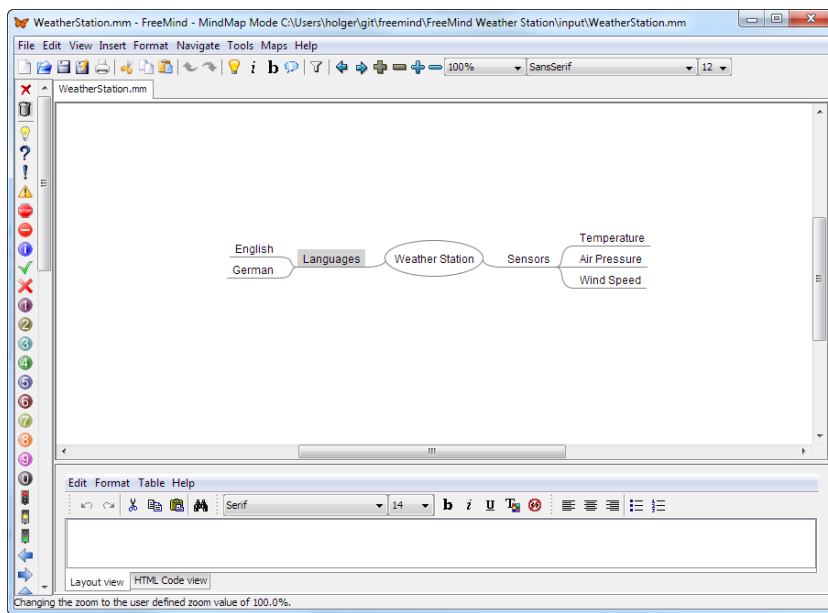
The next step for preparation is to import the FreeMind source code into Eclipse. Download the [source archive](#) from the FreeMind project at Sourceforge. Now go back to your Eclipse development environment and use option *File -> Import -> General -> Existing projects into Workspace*. Choose *Select archive file*, select the downloaded file and press *Finish*. After import, you should see the `Freemind_integration` project in your workspace.

To run our new project, we'll need a run configuration for FreeMind. Create a new *Java Application* and set the options as shown in the following figure. Please add also `-xmx256M` to the *VM arguments* field on the second page.

Figure 8. FreeMind launch configuration



Press the *Run* button to start the FreeMind tool. In FreeMind, open the example mind map located in the input folder of the FreeMind example project. You should now see the mind map as shown in the following figure.

Figure 9. FreeMind with example

As the last step we try to build the FreeMind distribution. Go back to Eclipse. In the Project Explorer view select `build.xml` located in the `freemind_integration` project. Open the context menu and choose *Run As -> Ant Build*. This will start the complete build process and copy all files into folder `bin/dist`. Please check the console output for the location on your disk. If there are problems, select *Run As -> 2 Ant Build ...*, and on the *JRE* tab make sure to have JRE 1.6 configured as *Separate JRE*.

To test the build, open a command line window and change the current directory to the `bin/dist/lib` folder. Start FreeMind with the following command:

```
java -jar freemind.jar freemind.main.FreeMindStarter
```

4. Level 1 Integration

The Level 1 integration will provide basic editing support inside the tool. It also provides interfaces to read the VEL description from the tool and process a VEL configuration inside the tool. We'll also implement a `pure::variants` transformation module. This will give a first impression of a integration within a very short time.

In this tutorial we'll restrict nodes of a mind map with restrictions. The result of such restrictions will be used to decide if a node is part of a variant or not. Because FreeMind supports user attributes on a node, we'll use a special attribute **pvRestriction** to store our restrictions.

4.1. Create `pure::variants` Restriction Action

To support the user during restriction editing we'll add an action into FreeMind which calls the `pure::variants` Desktop Hub by command line. The `pure::variants` installer places all needed executables into the `win` directory inside `pure::variants`' installation directory. The restriction editor is called like this:

```
%INSTALLDIR%/win/pvSCEditorRunner.exe "Restriction Code"
```

The editor will return the new restriction on standard output.

To add a new action into FreeMind we need to define it in the `freemind_actions.xsd` file located in the project root folder. Please add a new action `pv_node_action` at the beginning of the file.

```
.....
```

```
<xs:extension base="xml_action">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="pv_node_action"/>
    <xs:element ref="compound_action"/>
    <xs:element ref="select_node_action"/>
    .....
  </xs:choice>
</xs:extension>
```

Now define `pv_node_action` after `compound_action` element.

```
.....
</xs:element>
<!-- p::v restriction Action -->
  <xs:element name="pv_node_action">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="node_action">
          <xs:attribute name="value" use="optional" type="xs:string"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <!-- Select action. Selects a node. -->
  <xs:element name="select_node_action">
    .....
  </xs:element>
```

Now start the FreeMind runtime. This'll trigger a rebuild of the project. During this a `PvNodeAction` will be generated. We'll use this class later for our user interface action. Please make sure that the build finishes without any errors.

Add a new class into `PVAction.java` at the following location: `Freemind_integration/freemind/modes/mindmapmode/actions/`.

```
package freemind.modes.mindmapmode.actions;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import javax.swing.Action;
import javax.swing.JMenuItem;

import freemind.controller.MenuItemSelectedListener;
import freemind.controller.actions.generated.instance.PvNodeAction;
import freemind.controller.actions.generated.instance.XmlAction;
import freemind.modes.MindMap;
import freemind.modes.MindMapNode;
import freemind.modes.NodeAdapter;
import freemind.modes.attributes.Attribute;
import freemind.modes.mindmapmode.MindMapController;
import freemind.modes.mindmapmode.actions.xml.ActionPair;

public class PVAction extends NodeGeneralAction implements NodeActorXml,
  MenuItemSelectedListener {

  public static final String EDITOR      = "C:/Program Files/pure-systems/pv_Enterprise_4.0/
win/pvSCLEditorRunner.exe";
  public static final String PV_REST_ATTR = "pvRestriction";

  public PVAction(MindMapController controller) {
    super(controller, null, null);
    setName("pure::variants Restriction");
    addActor(this);
  }

  public Class getDoActionClass() {
    return PvNodeAction.class;
  }

  public boolean isSelected(JMenuItem checkItem, Action action) {
    return false;
  }
}
```

```

}

public ActionPair apply(MindMap model, MindMapNode selected) {
    ActionPair action = null;
    String oldvalue = selected.getAttribute(PV_REST_ATTR);
    String newvalue = enterRestriction(oldvalue);
    if (newvalue != null && newvalue != oldvalue){
        action = new ActionPair(createAction(selected, newvalue), createAction(selected,
oldvalue));
    }
    return action;
}

private PvNodeAction createAction(MindMapNode node, String value) {
    PvNodeAction action = new PvNodeAction();
    action.setNode(getNodeID(node));
    action.setValue(value);
    return action;
}

private String enterRestriction(String oldvalue) {
    StringBuilder sb = new StringBuilder();
    try {
        String[] args = new String[] { EDITOR, oldvalue == null ? "" : oldvalue };
        Process p = Runtime.getRuntime().exec(args);
        BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));
        String line = null;
        while ((line = in.readLine()) != null) {
            if (sb.length() > 0) {
                sb.append("\n");
            }
            sb.append(line);
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    return sb.toString();
}

public void act(XmlAction action) {
    if (action instanceof PvNodeAction) {
        PvNodeAction pva = (PvNodeAction) action;
        NodeAdapter node = getNodeFromID(pva.getNode());
        String value = pva.getValue();
        int row = node.getAttributePosition(PV_REST_ATTR);
        if (value == null) {
            if (row != -1) {
                node.getAttributes().removeRow(row);
            }
        }
        else {
            if (row == -1) {
                Attribute pvattr = new Attribute(PV_REST_ATTR, value);
                node.createAttributeTableModel();
                node.getAttributes().addRowNoUndo(pvattr);
            }
            else {
                node.getAttributes().setValue(row, value);
            }
        }
        modeController.nodeChanged(node);
    }
}
}

```

Please make sure that the path to the executable matches your environment (the hardcoded value of PVAction.EDITOR).

Finally add the new action into the mind map by editing the `MindMapController` class located in `Freemind_integration/freemind/modes/mindmapmode`. First add a member of type `PVAction` to the class.


```
public PVAction pvaction = null;
```

In the `createStandardActions` method assign a new instance to that member.

```
private void createStandardActions() {
    pvaction = new PVAction(this);
    ....
}
```

Finally, add our action to the enable/disable code in `setAllActions`.

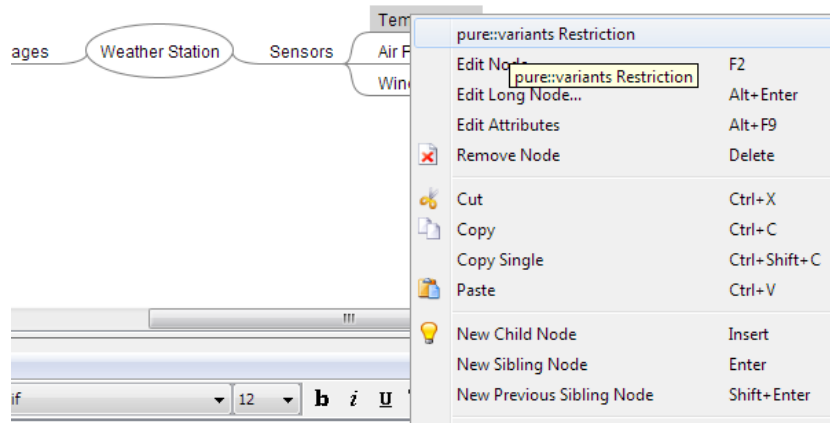
```
protected void setAllActions(boolean enabled) {
    logger.fine("setAllActions:" + enabled);
    super.setAllActions(enabled);
    // own actions
    pvaction.setEnabled(enabled);
    ....
}
```

In the `mindmap_menus.xml` located in the project root add `pvaction`.

```
....
<menu_category name="mindmapmode_popup">
  <menu_category name="edit">
    <menu_action field="pvaction"/>
    <menu_action name="edit" field="edit" key_ref="keystroke_edit"/>
  </menu_category>
</menu_category>
....
```

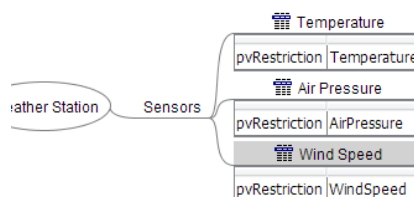
Start the FreeMind runtime and select a node inside the mind map. In the context menu there should be a new action `pure::variants Restriction`.

Figure 10. Restriction context menu action



After clicking the new action, a restriction editor will open and, after pressing `OK`, a new `pvRestriction` attribute will be added to the node.

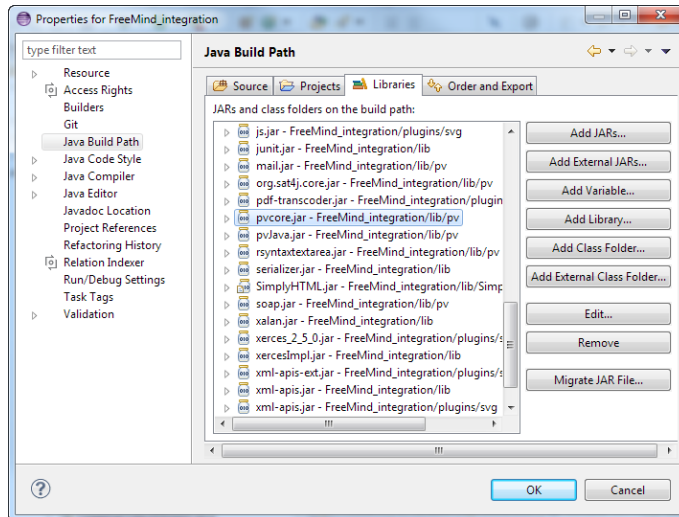
Figure 11. Added pvRestriction attributes



4.2. Write VEL Description

The next step is to write the variation points into a VEL description. The description can then be processed by a variant management tool. The pure::variants integration libraries contain abstractions to read and write VEL. Please copy all jars from the pure::variants installation subdirectory `jars` into the FreeMind project. Create a subfolder `pv` in the existing `lib` folder and copy the libraries into that folder. Then add all copied libraries to the project build path by using the project's *properties* dialog.

Figure 12. Add pure::variants libraries to the build path



We need also to add the libraries to the build.xml. Please create a new property `pvLib` in the build file and also append it as a value to the end of the `classpath` property (`:${pvLib}`).

```
....
<property name="xsltLib4" location="lib/xercesImpl.jar" />
<property name="pvLib" location="lib/pv/activation.jar:lib/pv/annotations.jar:lib/pv/
antlr-2.75.jar:lib/pv/autocompletable.jar:lib/pv/mail.jar:lib/pv/org.sat4j.core.jar:lib/
pv/pvcore.jar:lib/pv/pvJava.jar:lib/pv/rsyntaxtextarea.jar:lib/pv/soap.jar:lib/pv/
xpp3-1.1.4c.jar" />

<property name="classpath" value="${jibxruntimelibs}:${jibxjar}:${formsLib}:${simplyHtml}:
${spellLib}:${xsltLib}:${xsltLib2}:${xsltLib3}:${xsltLib4}:${pvLib}" />
<path id="jars.path">
....
```

We need also to add them to the file set to copy during distribution build.

```
....
<copy todir="${dist.lib}">
  <fileset dir="${src}/lib">
    <include name="pv/*.jar" />
    <include name="jorth.jar" />
    <include name="xalan.jar" />
    <include name="serializer.jar" />
  </fileset>
</copy>
....
```

Jump to manifest tag and add following attribute.

```
....
<attribute name="Class-Path" value="freemind.jar .. commons-lang-2.0.jar forms-1.0.5.jar
jibx/jibx-run.jar jibx/xpp3.jar jibx-run.jar xpp3.jar bindings.jar jorth.jar xalan.jar
serializer.jar xml-apis.jar xercesImpl.jar SimplyHTML/gnu-regexp-1.1.4.jar gnu-
regexp-1.1.4.jar SimplyHTML/SimplyHTML.jar SimplyHTML.jar pv/activation.jar pv/annotations.jar
pv/antlr-2.75.jar pc/autocompletable.jar pv/mail.jar pv/org.sat4j.core.jar pv/pvcore.jar pv/
pvJava.jar pv/rsyntaxtextarea.jar pv/soap.jar pv/xpp3-1.1.4c.jar" />
....
```

Now, create the class `VELExport.java` in the `freemind/main` package with the following code:

```
package freemind.main;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.InputStream;

import com.ps.consul.eclipse.vel.generated.ArtifactElementType;
import com.ps.consul.eclipse.vel.generated.CapabilityType;
import com.ps.consul.eclipse.vel.generated.ExpressionEnum;
import com.ps.consul.eclipse.vel.generated.ExpressionType;
import com.ps.consul.eclipse.vel.generated.ObjectFactory;
import com.ps.consul.eclipse.vel.generated.OptionalStructuralVariationpointType;
import com.ps.consul.eclipse.vel.generated.OptionalVariationType;
import com.ps.consul.eclipse.vel.generated.VariabilityApiEnum;
import com.ps.consul.eclipse.vel.generated.VariabilityExchangeModelType;
import com.ps.consul.eclipse.vel.generated.VariabilityExchangeModelsType;
import com.ps.consul.eclipse.vel.writer.VELWriter;
import com.ps.eclipse.core.runtime.CoreException;
import com.ps.eclipse.core.runtime.NullProgressMonitor;
import com.ps.xml.Tag;
import com.ps.xml.TagReader;
import com.ps.xml.TagVisitor;

class ToVELVisitor extends TagVisitor {
    private ObjectFactory factory = null;
    private VariabilityExchangeModelType velDescription = null;

    public ToVELVisitor(ObjectFactory f, VariabilityExchangeModelType desc) {
        factory = f;
        velDescription = desc;
    }

    public boolean visit(Tag t) {
        // we have a restriction on the node
        if (t.getName().equals("attribute") == true &&
            t.getAttributeValue("NAME").equals("pvRestriction")) {
            // get id and code
            Tag node = t.getParent();
            String nodeid = node.getAttributeValue("ID");
            String nodename = node.getAttributeValue("TEXT");
            // create optional variation point
            OptionalStructuralVariationpointType variationpoint =
            factory.createOptionalStructuralVariationpointType();
            variationpoint.setId("vp" + nodeid);
            variationpoint.setName(nodename);
            OptionalVariationType variation = factory.createOptionalVariationType();
            variation.setId("v" + nodeid);
            variation.setName(nodename);
            // add the restriction code
            ExpressionType expression = factory.createExpressionType();
            expression.setType(ExpressionEnum.PVSCL_EXPRESSION);
            expression.setValue(t.getAttributeValue("VALUE"));
            variation.setCondition(expression);
            variationpoint.getVariation().add(variation);
            velDescription.getVariationpointGroup().add(variationpoint);
            // add reference to the freemind node
            ArtifactElementType artifact = factory.createArtifactElementType();
            artifact.setUri("id://" + nodeid);
            variation.getCorrespondingVariableArtifactElement().add(artifact);
        }
        return super.visit(t);
    }
}

public class VELExport {

    public static void main(String[] args) {
        // first argument freemind file to read
        File freemindfile = new File(args[0]);
    }
}
```

```

// second argument vel file to write
File velfile = new File(args[1]);
try {
    export(freemindfile, velfile);
}
catch (Exception e) {
    e.printStackTrace();
}
}

public static void export(File freemindfile, File velfile) throws Exception {
    // read the freemind file
    TagReader reader = new TagReader();
    reader.read(freemindfile, new NullProgressMonitor());
    VariabilityExchangeModelsType velDocument = export(reader);
    // write VEL to disk
    VELWriter write = new VELWriter();
    write.writeVEL(velDocument, velfile);
}

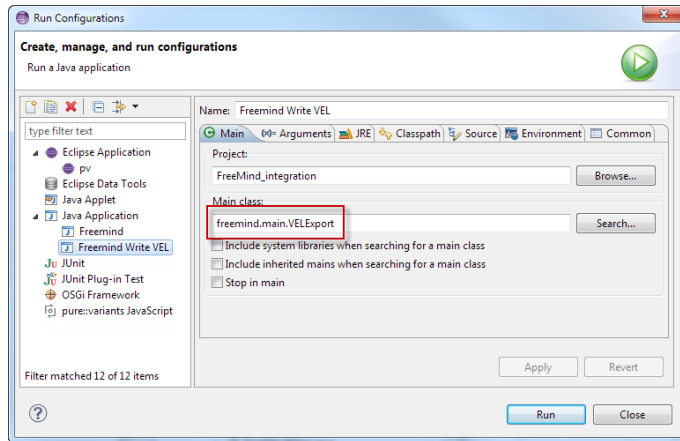
public static VariabilityExchangeModelsType export(String xml) throws Exception {
    VariabilityExchangeModelsType velDocument = null;
    if (xml != null && xml.length() > 0) {
        try {
            InputStream stream = new ByteArrayInputStream(xml.getBytes("UTF-8"));
            TagReader reader = new TagReader();
            reader.read(stream, new NullProgressMonitor());
            velDocument = export(reader);
        }
        catch (Exception ex) {}
    }
    return velDocument;
}

private static VariabilityExchangeModelsType export(TagReader reader) throws CoreException {
    VariabilityExchangeModelsType velDocument = null;
    Tag root = reader.getChild("node");
    if (root != null) {
        // get the name of the root node
        String name = root.getAttributeValue("TEXT");
        // create the VEL
        ObjectFactory factory = new ObjectFactory();
        velDocument = factory.createVariabilityExchangeModelsType();
        velDocument.setId("Doc" + name);
        velDocument.setVersion(1);
        // add capabilities
        CapabilityType capability = factory.createCapabilityType();
        capability.setImportVariabilityExchangeModel(false);
        capability.setExportVariabilityExchangeModel(true);
        capability.setGetConfiguration(false);
        capability.setSetConfiguration(true);
        velDocument.setCapability(capability);
        // create the description
        VariabilityExchangeModelType velDescription =
factory.createVariabilityExchangeModelType();
        velDescription.setId("FreeMind" + name);
        velDescription.setType(VariabilityApiEnum.VARIATIONPOINT_DESCRIPTION);
        velDescription.setName(name);
        velDocument.getVariabilityExchangeModel().add(velDescription);
        // iterate over the nodes and add variability information
        TagVisitor tv = new ToVELVisitor(factory, velDescription);
        reader.accept(tv);
    }
    return velDocument;
}
}

```

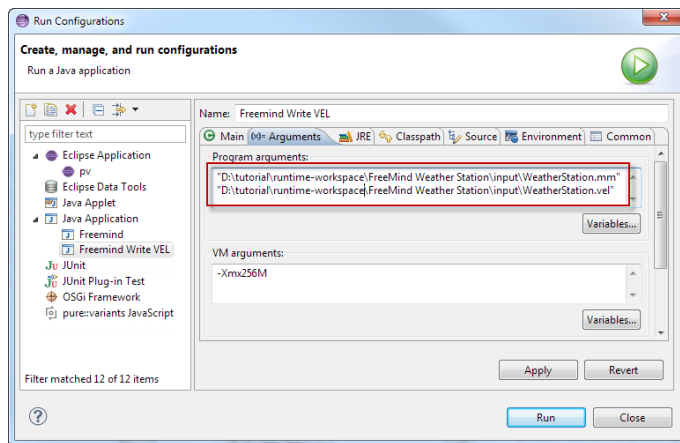
Run the VEL generation by creating a new launch configuration which uses the `VELExport` class to start.

Figure 13. Launch configuration for VEL export



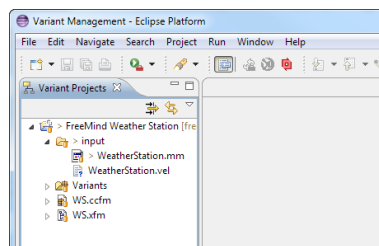
Set the first argument to the FreeMind mind map in your pure::variants runtime workspace. The second argument is the file to write. It should be placed in the same folder as the input file.

Figure 14. Arguments for VEL export



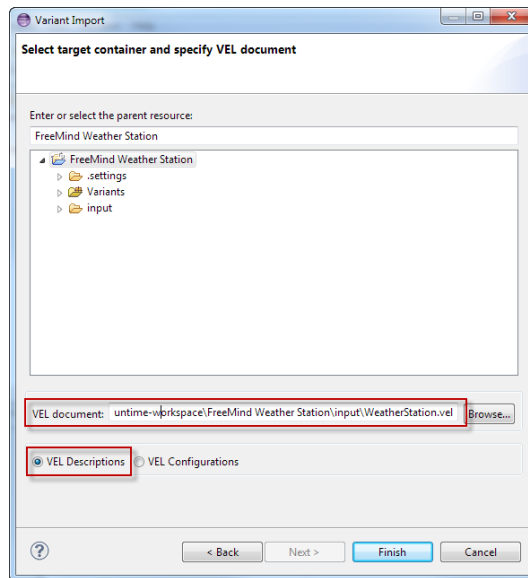
Pressing the Run button will start the export. Start the pure::variants runtime now. We should find a new file next to the mind map.

Figure 15. Arguments for VEL export



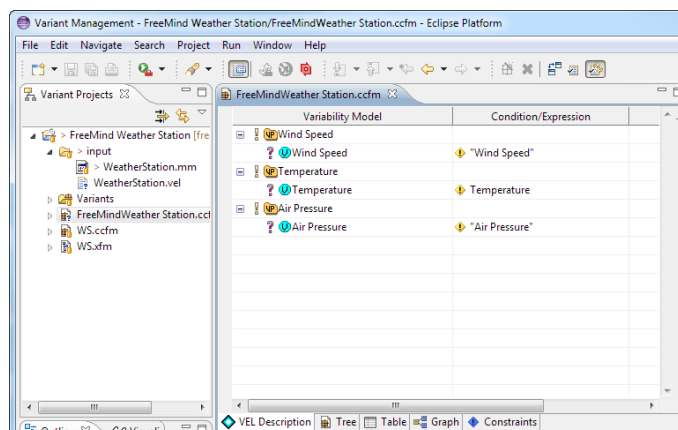
The VEL file can now be imported into a pure::variants family model. Start the import with *File -> Import -> Variant Models or Projects*. Select *Import from VEL document* from the next page. Enter the location for your VEL file in the *VEL document* field. Set mode to *VEL Descriptions* and press *Finish*.

Figure 16. VEL Description Import



The import will create a new family model (*FreemindWeather Station.cfm*) in the project. It contains all variation points of the FreeMind mind map.

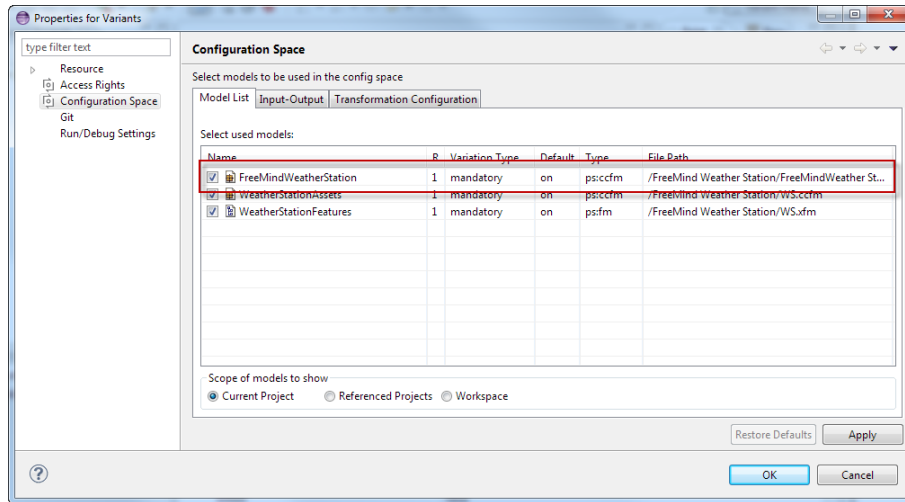
Figure 17. VEL Description Family Model



4.3. Process VEL Configuration

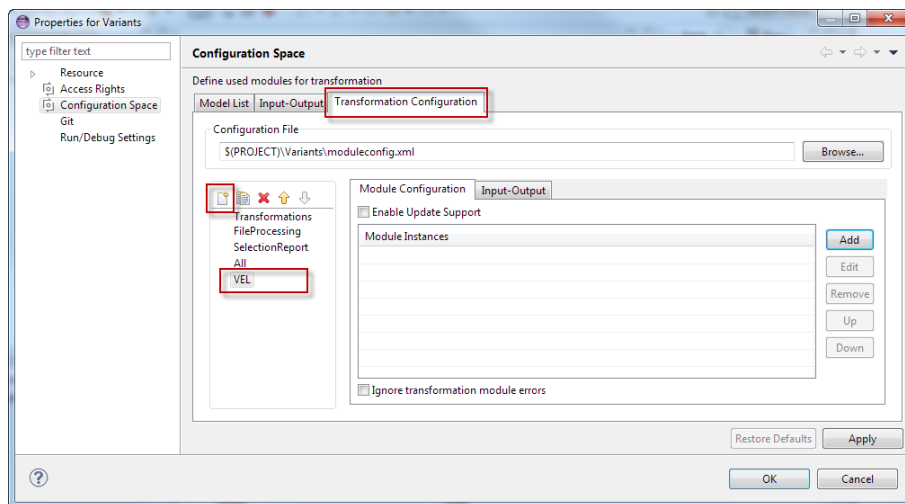
To create a VEL configuration we'll use the VEL transformation module. First add the imported variation point model to the configuration space. Open the properties dialog on the *configuration space* and check the *FreeMindWeatherStation* model.

Figure 18. Add VEL Description model to Configuration Space



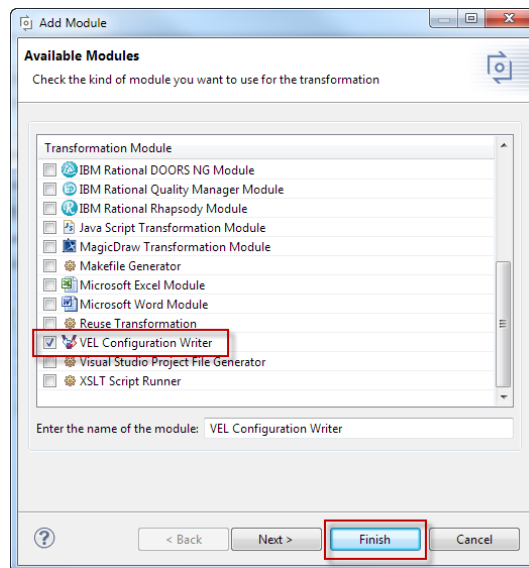
Then go to the *Transformation Configuration* tab and create a new transformation called *VEL*.

Figure 19. Create VEL transformation configuration



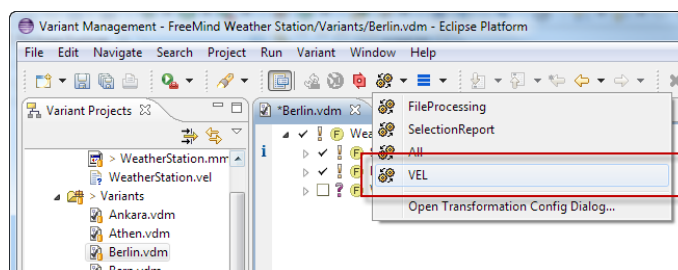
Add a new module instance by pressing the *Add* button. From the list of available modules select the *VEL Configuration Writer* and press *Finish*.

Figure 20. Select VEL configuration writer



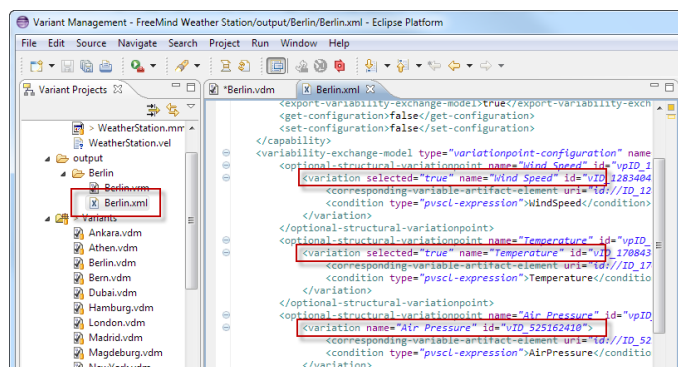
Close the configuration properties dialog by pressing *OK*. Now open a *variant description model* and start the *VEL* transformation.

Figure 21. Start VEL transformation



After finishing the transformation, you'll find the VEL configuration file in the output folder. Depending on your selection some variation points will be selected.

Figure 22. VEL configuration



We can now process this VEL configuration inside FreeMind and create a variant of the mind map by removing all nodes that are not selected. Create a new class `VELConfig.java` in the `freemind/main` package with the following code:

```
package freemind.main;

import java.io.File;
import java.util.HashSet;
```



```

import java.util.Iterator;
import java.util.Set;

import com.ps.consul.eclipse.vel.VELVisitor;
import com.ps.consul.eclipse.vel.generated.ArtifactElementType;
import com.ps.consul.eclipse.vel.generated.VariabilityApiEnum;
import com.ps.consul.eclipse.vel.generated.VariabilityExchangeModelType;
import com.ps.consul.eclipse.vel.generated.VariabilityExchangeModelsType;
import com.ps.consul.eclipse.vel.generated.VariationType;
import com.ps.consul.eclipse.vel.reader.VELReader;
import com.ps.eclipse.core.runtime.CoreException;
import com.ps.eclipse.core.runtime.NullProgressMonitor;
import com.ps.xml.Tag;
import com.ps.xml.TagReader;

public class VELConfig {

    public static void main(String[] args) {
        File infile = new File(args[0]);
        File outfile = new File(args[1]);
        File velfile = new File(args[2]);
        try {
            config(infile, outfile, velfile);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void config(File in, File out, File vel) throws Exception {
        // read the VEL configuration and return nodes to remove
        Set removeids = scanVEL(vel);
        // read the mind map
        TagReader reader = new TagReader();
        reader.read(in, new NullProgressMonitor());
        // filter the nodes
        filterNodes(removeids, reader);
        // write the output file
        reader.write(out, new NullProgressMonitor());
    }

    public static void filterNodes(Set removeids, Tag node) {
        Iterator idx = node.getChildren().iterator();
        while (idx.hasNext()) {
            Tag child = (Tag) idx.next();
            String id = null;
            if (child.hasAttribute("ID") == true) {
                id = child.getAttributeValue("ID");
            }
            if (id != null && removeids.contains(id) == true) {
                idx.remove();
            }
            else {
                filterNodes(removeids, child);
            }
        }
    }

    public static Set scanVEL(File vel) throws CoreException {
        VELReader velread = new VELReader();
        VariabilityExchangeModelsType velDocument = velread.readVEL(vel);
        return scanVEL(velDocument);
    }

    public static Set scanVEL(VariabilityExchangeModelsType velDocument) {
        final Set toremove = new HashSet();
        VELVisitor visitor = new VELVisitor() {

            protected int show(VariabilityExchangeModelType model) {
                return model.getType() == VariabilityApiEnum.VARIATIONPOINT_CONFIGURATION ?
                VISIT_CHILDS : SKIP_CHILDS;
            }
        }
    }
}

```

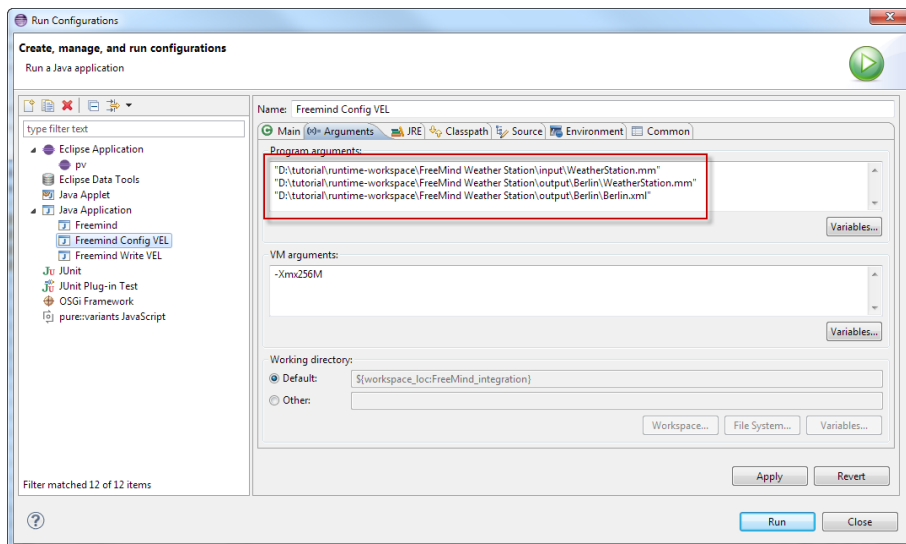
```

protected int show(VariationType variation) {
    if (variation.isSelected() == null || variation.isSelected().booleanValue() == false)
    {
        Iterator adx = variation.getCorrespondingVariableArtifactElement().iterator();
        while (adx.hasNext()) {
            ArtifactElementType artifact = (ArtifactElementType) adx.next();
            toremove.add(artifact.getUri().substring("id://".length()));
        }
    }
    return SKIP_CHILDSD;
}
};
visitor.accept(velDocument);
return toremove;
}
}

```

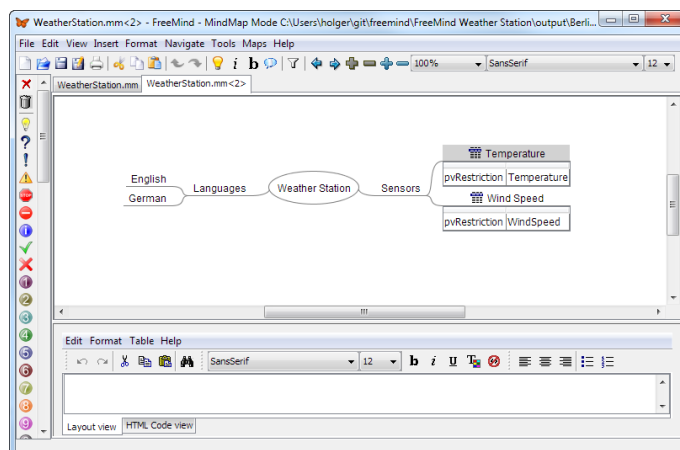
We can now run the variant generation, by creating a new launch configuration which uses the `VELConfig` class to start. We need three arguments, the input file, the output file and the VEL configuration to process.

Figure 23. Run FreeMind variant generation



After finishing the generation, we have a new FreeMind file in the output folder of the runtime workspace. The generated mind map contains only restricted nodes with references to selected features.

Figure 24. Generated Mind Map Variant

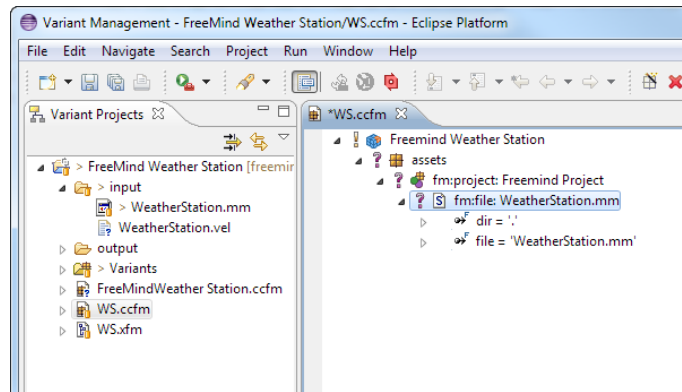


Before we can continue and create a pure::variants transformation module, we need to rebuild the FreeMind distribution. Select the `build.xml` in the project view. Open the context menu and choose *Run As -> Ant Build*.

4.4. Create pure::variants Transformation Module

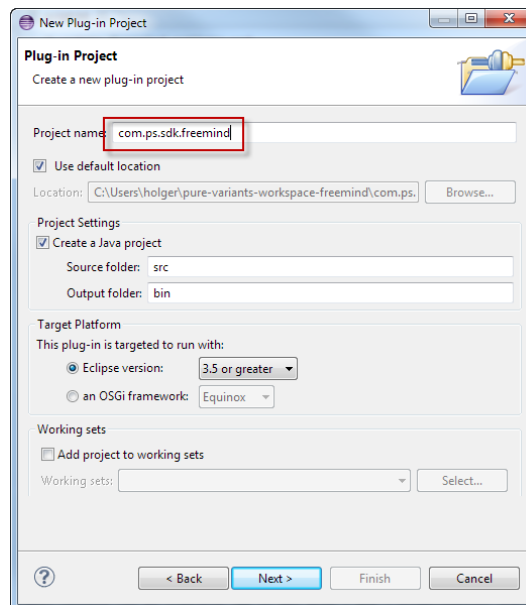
Before we start creating our transformation module we'll need a representation of the FreeMind files inside a Family Model. The Family Model element will contain the path to the FreeMind file. The following model structure is used by most pure::variants connectors. A part will act as a project folder. It contains a list of project files. For FreeMind we introduce the `fm:project` type for the part element and the `fm:file` type for the source element. The file element has the attributes `dir` and `file` to point to the mind map file. Start the pure::variants runtime. Open the `WS.ccfm` and add the FreeMind family model elements to that model.

Figure 25. FreeMind Family Model



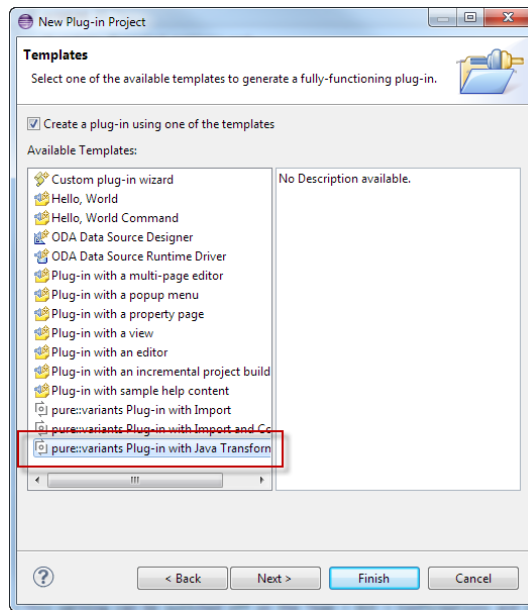
Close the runtime and create a new Plugin-Project `com.ps.sdk.freemind`.

Figure 26. New Plugin Project



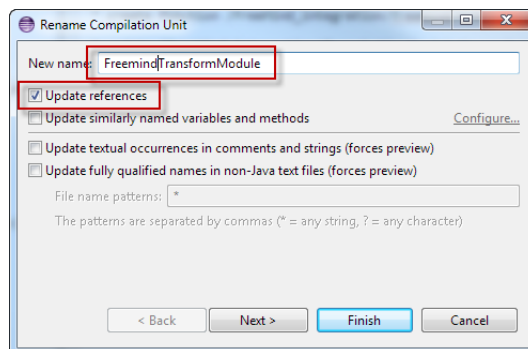
Press *Next* two times and select *pure::variants Plug-in with Java Transformation* on the template page.

Figure 27. Select Template



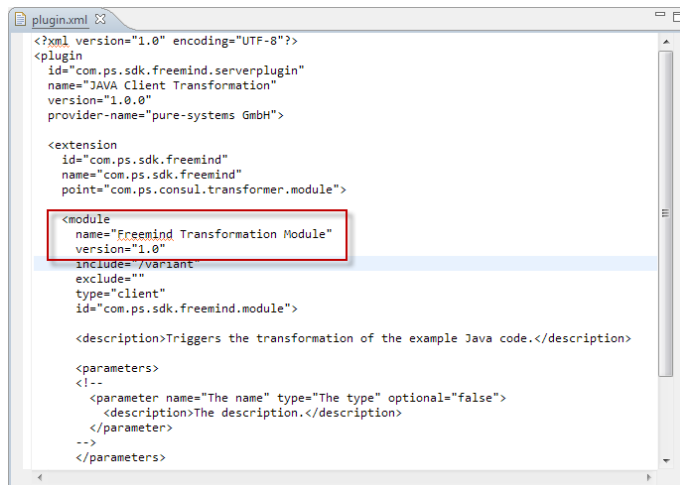
Press *Finish* to create the project. The wizard creates the basic project structure for a Java based transformation module. As the first step we'll rename the `JAVAClientTransformationModule` class to `FreemindTransformationModule`. Please use the *Refactor -> Rename* action in the project view. Ensure the Update references check mark is activated.

Figure 28. Rename Transformation Module Class



Next we open the `plugin.xml` in the `server/clientmodule` folder and change the module name to *FreeMind Transformation Module*. This text will be shown later in the module selection dialog when creating a transformation configuration.

Figure 29. Rename Transformation Module

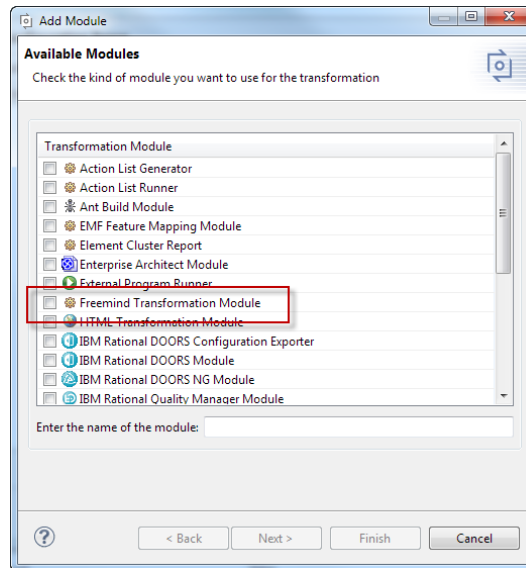


If you now start the pure::variants runtime and open the module selection dialog, you should see the new transformation module in the list of available modules.

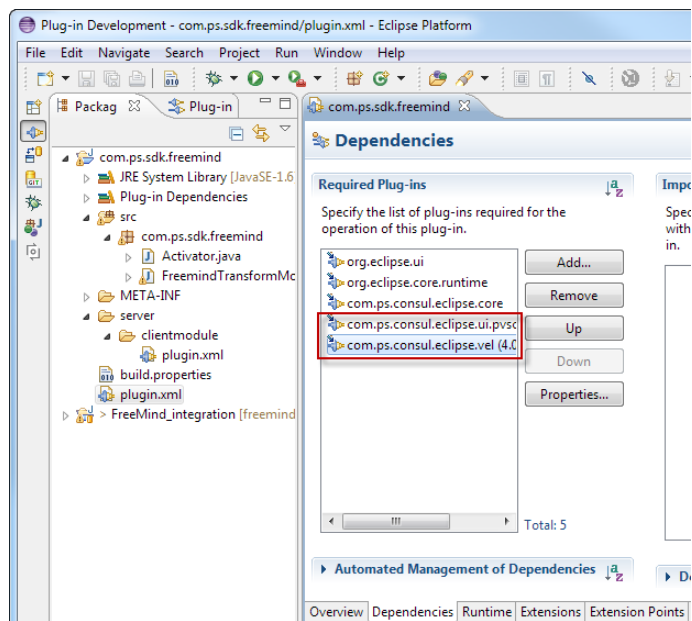
Tip

If your changes in the plugin.xml of the server client module are not visible in the runtime, please close the pure::variants Desktop Hub through its context-menu in the Windows taskbar. If the editor uses the same workspace it will cache data and your changes will not be visible.

Figure 30. FreeMind Transformation Module



Our transformation module needs additional dependencies to standard pure::variants plugins. Open the plugin.xml in the project root folder and add `com.ps.consul.eclipse.ui.pvseditor` and `com.ps.consul.eclipse.vel` to the Dependencies.

Figure 31. Add Plugin Dependencies

Now we'll start updating the transformation module. Add the following member variables.

```
private ReferenceStore m_RefStore=null;
private Evaluator m_Evaluator = null;
```

In the `work` method we collect all FreeMind files and transform them.

```
public ClientTransformStatus work(IProgressMonitor monitor) {
    ClientTransformStatus status = new ClientTransformStatus(ClientTransformStatus.OK, "");
    try {
        m_RefStore = new ReferenceStore(getModels());
        m_Evaluator = new Evaluator(getVariantEnvironment());
        m_Evaluator.registerVerifier(new PVSCLVerifier());

        FilesCollector fc = new FilesCollector(ModelConstants.PART_CLASS,
            "fm:project", "fm:file", null);
        List<FileElement> files = fc.collect(getModels());
        for(FileElement fe : files) {
            File inputfile = fe.getSourcePath(getInput());
            File outputfile = fe.getDestinationPath(getOutput());
            if (inputfile.exists() == false) {
                throw new FileNotFoundException("File not found: " + inputfile.getAbsolutePath());
            }
            File outputdir = outputfile.getParentFile();
            if( outputdir.exists() == false ) {
                if( outputdir.mkdirs() == false ) {
                    throw new IOException("Could not create directory: " + outputdir.getAbsolutePath());
                }
            }
            transform(inputfile,outputfile);
        }
    } catch (Exception e) {
        status.setStatus(ClientTransformStatus.ERROR);
        status.setMessage(e.getMessage());
    }
    return status;
}
```

The transform is split into three steps per file. First, call the FreeMind VEL export to generate the VEL description of the given file. In the next step, the VEL configuration will be created. The last step calls the FreeMind VEL configuration processing and saves the resulting mind map. Please *update* the path to the FreeMind distribution directory so it matches your environment (see constant `FREEMIND_LIB`).

```

private static final String FREEMIND_LIB = "D:/tutorial/bin/dist/lib/freemind.jar";

private void transform(File inputfile, File outputfile) throws Exception {
    File veldesc = File.createTempFile("vel_desc", null);
    File velconfig = File.createTempFile("vel_config", null);
    createVelDescription(inputfile, veldesc);
    createVelConfig(veldesc, velconfig);
    createFreemindVariant(inputfile, outputfile, velconfig);
}

private void createVelDescription(File inputfile, File veldesc) throws IOException,
InterruptedException {
    List<String> cmd = new ArrayList<String>();
    cmd.add("java");
    cmd.add("-cp");
    cmd.add(FREEMIND_LIB);
    cmd.add("freemind.main.VELExport");
    cmd.add(inputfile.getAbsolutePath());
    cmd.add(veldesc.getAbsolutePath());
    String[] cmds = cmd.toArray(new String[cmd.size()]);
    Process proc = Runtime.getRuntime().exec(cmds);
    proc.waitFor();
}

private void createVelConfig(File veldesc, File velconfig) throws CoreException {
    List<IPVModel> descmodels = loadVEL(veldesc);
    List<IPVModel> configmodels = m_Evaluator.evaluate(m_RefStore, descmodels);
    writeVEL(descmodels, configmodels, velconfig);
}

private void createFreemindVariant(File inputfile, File outputfile, File velconfig) throws
IOException, InterruptedException {
    List<String> cmd = new ArrayList<String>();
    cmd.add("java");
    cmd.add("-cp");
    cmd.add(FREEMIND_LIB);
    cmd.add("freemind.main.VELConfig");
    cmd.add(inputfile.getAbsolutePath());
    cmd.add(outputfile.getAbsolutePath());
    cmd.add(velconfig.getAbsolutePath());
    String[] cmds = cmd.toArray(new String[cmd.size()]);
    Process proc = Runtime.getRuntime().exec(cmds);
    proc.waitFor();
}

```

The loading and writing of the VEL models needs the following helper methods.

```

private List<IPVModel> loadVEL(File veldesc) throws CoreException {
    List<IPVModel> models = new ArrayList<IPVModel>();
    VELDescriptionImporter vimporter = new
    VELDescriptionImporter(VELConditionRelation.ASSURANCE);
    VELReader vread = new VELReader();
    VariabilityExchangeModelsType vmmodel = vread.readVEL(veldesc);
    IPVFamilyModel[] ccfms = vimporter.importVELDescription(vmmodel);
    models.addAll(Arrays.asList(ccfms));
    return models;
}

private void writeVEL(List<IPVModel> descmodels, List<IPVModel> configmodels,
    File velconfig) throws CoreException {
    VELDescriptionExporter descriptionExporter = new VELDescriptionExporter();
    VariabilityExchangeModelsType velDocument =
    descriptionExporter.exportVELDescription(toFamilyArray(descmodels));
    VELConfigurationExporter vexp = new VELConfigurationExporter();
    vexp.exportVELConfiguration(toFamilyArray(configmodels), velDocument);
    VELWriter vwrite = new VELWriter();
    vwrite.writeVEL(velDocument, velconfig);
}

protected IPVFamilyModel[] toFamilyArray(List<IPVModel> models) {
    IPVFamilyModel[] fams = new IPVFamilyModel[models.size()];
}

```

```

for (int i = 0; i < models.size(); ++i) {
    fams[i] = (IPVFamilyModel) models.get(i);
}
return fams;
}

```

The transformation can now be started by creating a new transformation configuration in the pure::variants runtime. Add the new *FreeMind Transformation Module* to the list of modules. Running the transformation will create the modified variant of the FreeMind mind map.

5. Level 2 Integration

The Level 2 integration will be integrated deeper into the tool. It will embed an integration user interface that allows loading pure::variants models, and will call the pure::variants restriction editor directly, without the need to use the external pure::variants Desktop Hub. Furthermore, it will provide a preview option, which grays out all restricted FreeMind nodes that would be removed during a transformation with the selected variant model. This preview will be based on the same VEL interfaces, already used for the transformation module.

5.1. Embed a pure::variants Integration User Interface

As the first step, we will embed the pure::variants integration user interface. Create a new package `pvintegration` in the `freemind` package and add class `PVFreeMindUserInterface` with the following code:

```

package freemind.pvintegration;

import freemind.main.FreeMind;
import tools.ToolIntegration;
import tools.properties.PropertyKeys;
import ui.mainui.UserInterface;

public class PVFreeMindUserInterface extends UserInterface {
    private FreeMind m_FreeMind;

    public void setFreeMind(FreeMind fm) {
        m_FreeMind = fm;
    }

    public void initialize(){
        ToolIntegration integration = ToolIntegration.getInstance();
        integration.init("pvFreeMind", "pure::variants Integration for FreeMind");
        integration.getPropertySerializer().addUserLevelKey(PropertyKeys.MODEL_HISTORY_KEY);
        integration.getPropertySerializer().addToolLevelKey(PropertyKeys.LASTLOADED_KEY);

        loadSavedModelProperties();
        integration.setIntegrationUI(this);
    }
}

```

Now we still need to add the user interface to FreeMind. To do this, go to `freemind.main.FreeMind`, add a new method `addPVIntegrationUI` as below, and insert the missing import statements.

```

private void addPVIntegrationUI(Container fmView) {
    PVFreeMindUserInterface pvView = new PVFreeMindUserInterface();
    pvView.setFreeMind(this);

    pvView.initialize();

    JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, pvView, fmView);
    splitPane.setOneTouchExpandable(true);
    splitPane.setContinuousLayout(true);
    splitPane.setAlignmentX(Component.CENTER_ALIGNMENT);
    splitPane.setDividerLocation(200);

    getContentPane().add(splitPane, BorderLayout.CENTER);
}

```



```
}
```

Now, go to `setScreenBounds()` and replace the following line

```
getContentPane().add(mTabbedPane, BorderLayout.CENTER);
```

with

```
addPVIntegrationUI(mTabbedPane);
```

Then, go to `setContentComponent()` and replace

```
getContentPane().add(mContentComponent, BorderLayout.CENTER);
```

with

```
addPVIntegrationUI(mContentComponent);
```

Finally, update the `windowClosing` event of method `setScreenBounds` as follows.

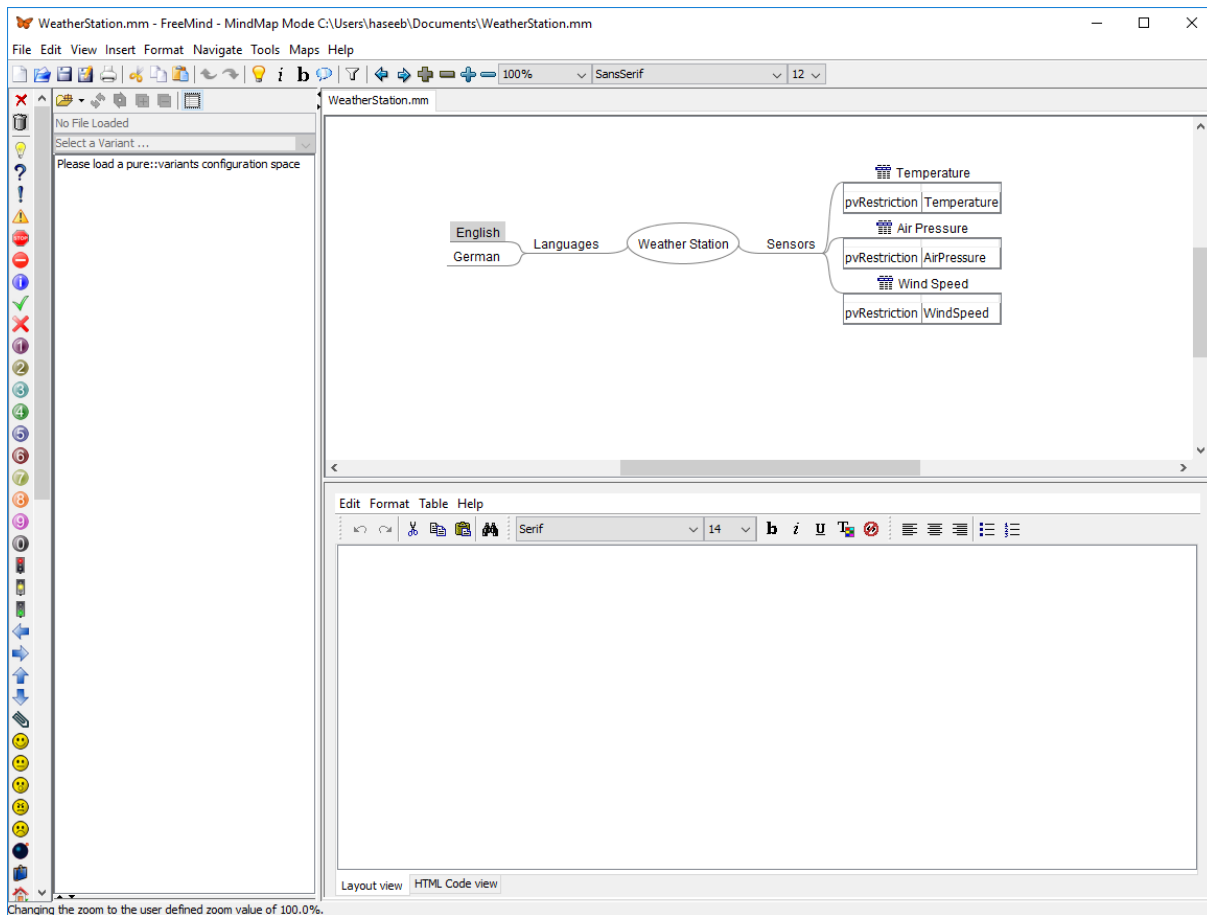
```
....
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {

        PVFreeMindUserInterface ui = null;
        try {
            // save all properties and dispose
            ui = ((PVFreeMindUserInterface)
ToolIntegration.getInstance().getCurrentPVUserInterface());
            ui.saveGlobalModelProperties();
            ui.saveLocalModelProperties();
            ToolIntegration.getInstance().dispose();
        }
        catch (Exception ex) {
            com.ps.object.Logger.log(ex);
        }

        controller.quit.actionPerformed(new ActionEvent(this, 0, "quit"));

        // if the user cancelled the close action, we need to reinitialize the
        // tool integration
        ToolIntegration.getInstance().init("pvFreeMind", "pure::variants Integration for
FreeMind");
        if (ui != null) {
            ui.loadSavedModelProperties();
        }
    }
}
....
```

Run the `FreeMind` configuration. You should see a *pure::variants* integrated view on the left side.

Figure 32. Pure-Variants Integrated View

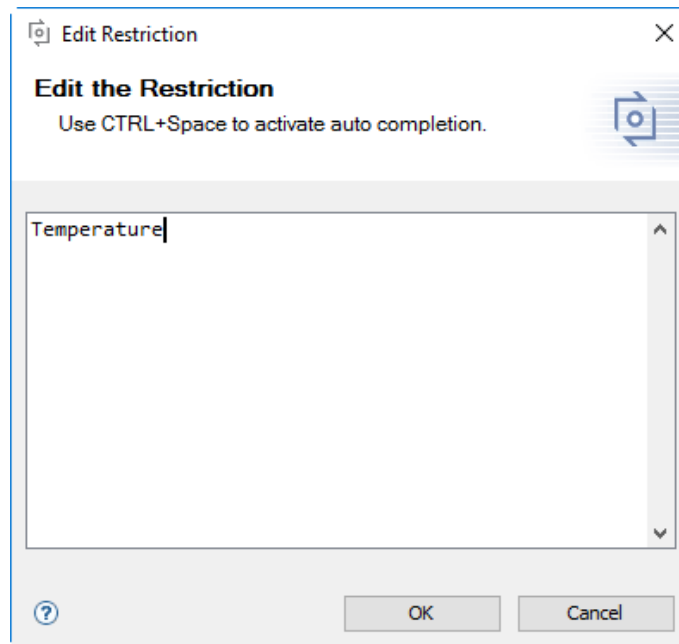
5.2. pure::variants Restriction Action

Next, we will replace the code for opening the pure::variants restriction editor. To do this, open `freemind.modes.mindmapmode.actions.PVAction` and replace the `enterRestriction` implementation with the following code.

```
private String enterRestriction(String oldvalue) {
    PVFreeMindUserInterface ui = ((PVFreeMindUserInterface)
ToolIntegration.getInstance().getCurrentPVUserInterface());
    PVSCLEditor editor = new PVSCLBooleanEditor("Restriction", ui.getJFrame(), oldvalue,
ui.getModelManager().getModels());
    editor.showDialog();

    String result = editor.getResult();
    if (result != null) {
        ui.saveLocalModelProperties();
    }
    return result;
}
```

Run the FreeMind configuration. From the context menu of any node, select *pure::variants Restriction*. The same editor as in the Level 1 integration should open. However, this time the Desktop Hub is not used.

Figure 33. PVSCL Restriction Window

5.3. Preview

The final task for Level 2 integration is to add a preview option to the pure::variants integration. The preview will gray out all restricted FreeMind nodes that would be removed during a transformation with the selected variant model. To do that, we first need to add a button, which triggers the preview.

First, open `PVFreeMindUserInterface` and add the following code. It contains a new member variable and a new constructor and method, which create the preview button. For convenience, we use an existing FreeMind icon. Alternatively, you can also add your own icon to the `images/icons/` directory and use that instead.

```
....
private FreeMind      m_FreeMind;
private JToggleButton m_PreviewButton = null;

public PVFreeMindUserInterface() {
    super();
    addVisualizationButtons();
}

protected void addVisualizationButtons() {
    ImageIcon icon = null;
    try {
        icon = new ImageIcon(this.getClass().getResource("../images/icons/xmag.png"));
    }
    catch (Exception ex) {
        Logger.log(ex);
    }

    m_PreviewButton = new JToggleButton(null, icon);
    m_PreviewButton.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {
            // TODO
        }

    });
    int index = m_Toolbar.getComponentCount() - 2;
    JToolBar.Separator s = new JToolBar.Separator(null);
    m_PreviewButton.setToolTipText("Preview");
    m_Toolbar.add(m_PreviewButton, index);
    m_Toolbar.add(s, index);
}
```

```

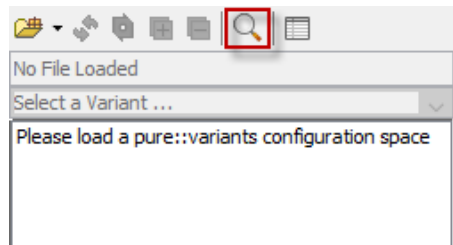
}

public void setFreeMind(FreeMind fm) {
....

```

Run the `Freemind` configuration. You should see a **Preview** button inside `pure::variants` view.

Figure 34. Preview button



Now, we'll make the **Preview** button functional. For that we need to make some changes on the `FreeMind` API, which enable setting and resetting the node's text color.

First, go to `freemind.modes.MindIcon` and make the constructor public.

```

....
public MindIcon(String name) {
    setName(name);
    associatedIcon = null;
}
....

```

Next, open class `freemind.modes.NodeAdapter` and add a private member variable of type `Color`

```

....
private static boolean          sSaveOnlyIntrinsicallyNeededIds      =
false;
private Color                   m_PvColor                          = null;

//
// Constructors
//
....

```

Then, add a new method `setPvColor`

```

public void setPvColor(Color color) {
    m_PvColor = color;
}

```

Replace the `getColor` method with

```

/** The Foreground/Font Color */
public Color getColor() {
    if (m_PvColor == null) {
        return color;
    }
    else {
        return m_PvColor;
    }
}

```

And replace the `getIcons` method with the following code. This will lighten all custom node icons, so they also appear as grayed out.

```

public List getIcons() {

```

```

if (icons == null) return Collections.EMPTY_LIST;
if (m_PvColor == null) {
    return icons;
}
else {
    // lighten all custom node icons, so they also appear grayed out
    Vector lightIcons = new Vector();
    for (Object mc : icons) {
        MindIcon mic = (MindIcon) mc;
        Image img = new Image(mic.getIcon().getImage());
        img = UiPlugin.getDefault().getImageManager().lightenImage(img, 50);
        img.setDescription(mic.getIcon().getDescription());
        MindIcon mic2 = new MindIcon(mic.getDescription(), img);
        lightIcons.add(mic2);
    }
    return lightIcons;
}
}
}

```

Now, open PVFreeMindUserInterface and add two new member variables of type Set and Color

```

....
private JToggleButton m_PreviewButton = null;
private Color         m_PvColor       = new Color(128, 128, 128);
private Set           m_RestrictedNodes = null;

public PVFreeMindUserInterface() {
....

```

Then, add a new method updateMindMap.

```

private void updateMindMap() {
    StringWriter sw = new StringWriter();
    VariabilityExchangeModelsType velDescription = null;
    try {
        MindMap map = m_FreeMind.getController().getMap();
        map.getXml(sw);
        velDescription = VELEExport.export(sw.toString());
        VariabilityExchangeModelsType velConfiguration = exportVELConfiguration(velDescription);
        m_RestrictedNodes = VELConfig.scanVEL(velConfiguration);
        for (Object nodeID : m_RestrictedNodes) {
            MindMapNode node = map.getLinkRegistry().getTargetForId(nodeID.toString());
            if (m_PreviewButton.isSelected()) {
                ((NodeAdapter) node).setPvColor(m_PvColor);
            }
            else {
                ((NodeAdapter) node).setPvColor(null);
            }
            map.nodeRefresh(node);
        }
    }
    catch (IOException e2) {
        com.ps.object.Logger.log(e2);
    }
    catch (Exception e1) {
        com.ps.object.Logger.log(e1);
    }
}
}

```

Finally, replace the //TODO in the button-click event of addVisualizationButtons() with a call to updateMindMap.

```

....
m_PreviewButton.addActionListener(new ActionListener() {

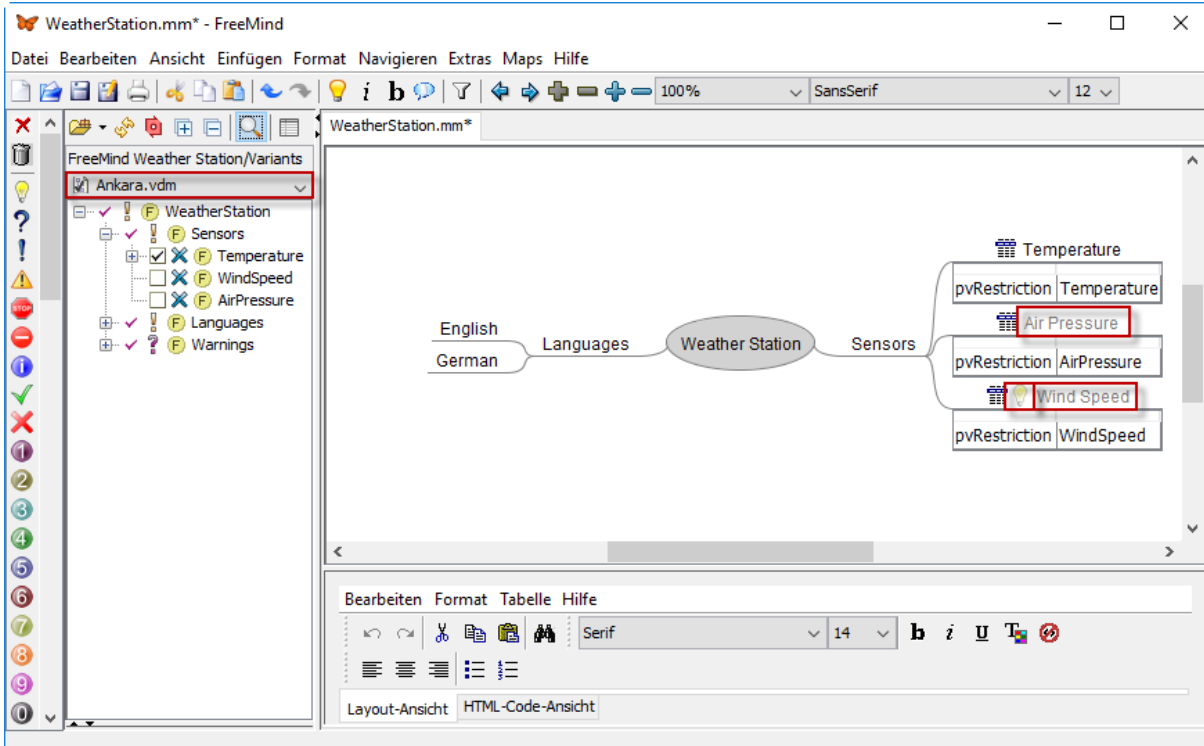
    public void actionPerformed(ActionEvent e) {
        updateMindMap();
    }

});

```

To test the preview, run the Freemind configuration. Load *configspace.xml* from *Variants* directory. Select a VDM from the drop-down box, and click the **Preview** button. The text of all restricted nodes should be grayed out. If you add any custom icons to the restricted nodes, they will also be lightened (e.g., the light bulb of the WindSpeed node). Toggle the button, and the node's text and icons will have the standard color again.

Figure 35. Transformation preview on FreeMind



As a final step, we will improve the preview, so that it is updated automatically when another variant is selected. For this we will handle pure::variants model events. Additionally, we only enable the preview button if a variant model is opened.

To do this, update the constructor of *PVFreeMindUserInterface*, and add a new method *resetNodes*:

```
import eventhandling.EventManager;
import eventhandling.IEventListener;
import eventhandling.events.Event;
import eventhandling.events.ModelEvent;
import eventhandling.events.ScopeEvent;
....
public PVFreeMindUserInterface() {
    super();
    addVisualizationButtons();

    EventManager eventManager = new EventManager();
    eventManager.attach(getModelManager());
    eventManager.attach(getModelLoader());

    eventManager.addListener(new IEventListener() {

        public void handleEvent(Event e) {
            if (e.getType() == ModelEvent.MODEL_OPENED && getModelManager().isVariantModelLoaded()
== true) {
                m_PreviewButton.setEnabled(true);
                if (m_PreviewButton.isSelected()) {
                    resetNodes();
                    updateMindMap();
                }
            }
        }
    });
}
```

```

    }
    else if (e instanceof ScopeEvent && getModelManager().isVariantModelLoaded() == false)
    {
        if (m_PreviewButton.isEnabled() == true) {
            m_PreviewButton.setEnabled(false);
            resetNodes();
        }
        m_PreviewButton.setSelected(false);
    }
    }
    });
}

private void resetNodes() {
    MindMap map = m_FreeMind.getController().getMap();
    if (m_RestrictedNodes != null) {
        for (Object nodeID : m_RestrictedNodes) {
            MindMapNode node = map.getLinkRegistry().getTargetForId(nodeID.toString());
            ((NodeAdapter) node).setPvColor(null);
            map.nodeRefresh(node);
        }
    }
}
}
....

```

Then, disable the **Preview** button at the start by updating `addVisualizationButtons()`.

```

....
    JToolBar.Separator s = new JToolBar.Separator(null);
    m_PreviewButton.setEnabled(false);
    m_PreviewButton.setToolTipText("Preview");
....

```

Run the `Freemind` configuration. When you switch between different variants, the preview should update automatically if the preview button is active.

