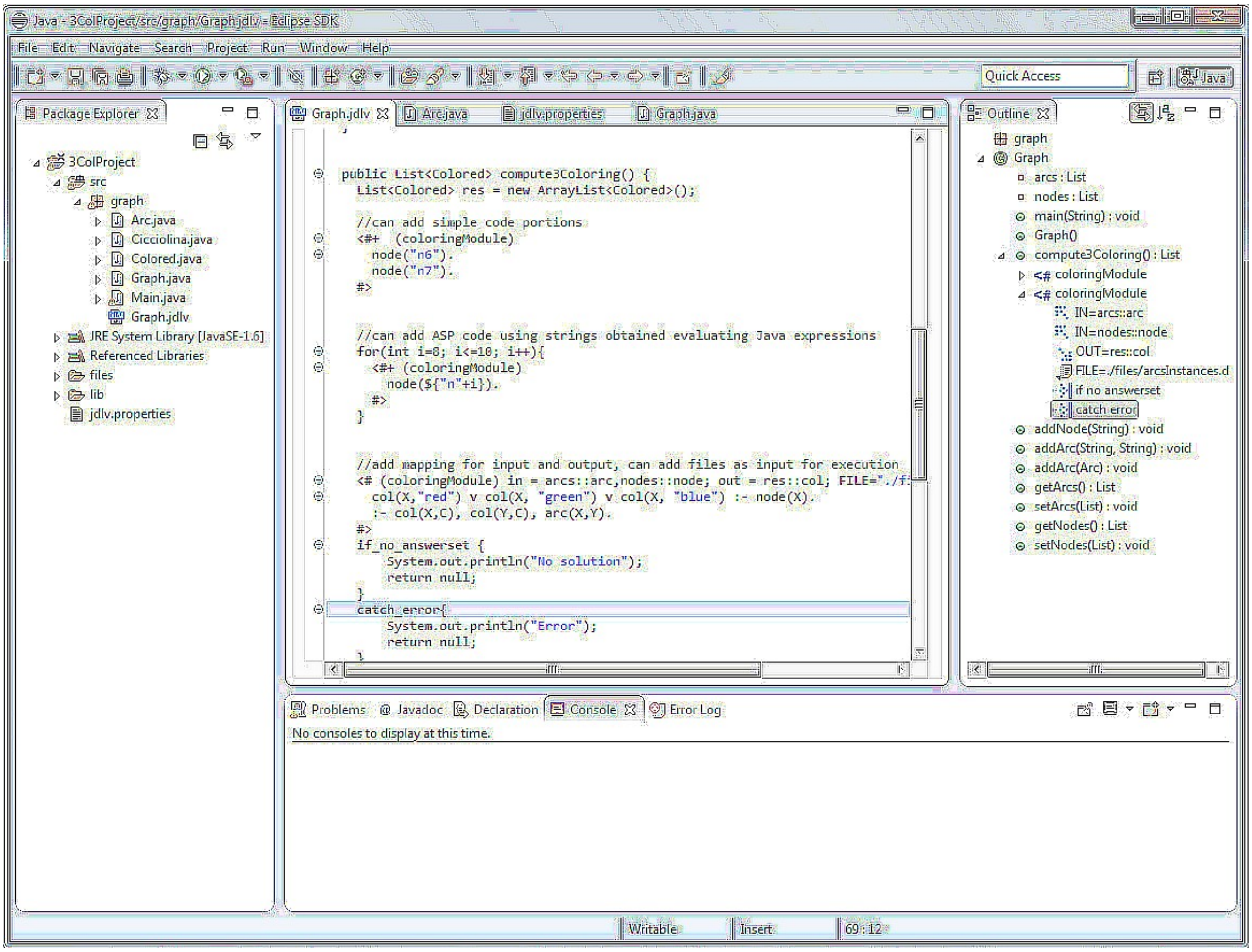
JDLV Eclipse plug-in



```
public List<Colored> compute3Coloring() {
  List<Colored> res = new ArrayList<Colored>();

  //can add simple code portions
  <#+  (coloringModule)
    node("n6").
    node("n7").
  #>


  //can add ASP code using strings obtained evaluating Java expressions
  for(int i=8; i<=10; i++){
    <#+ (coloringModule)
      node(${"n"+i}).
    #>
  }


  //add mapping for input and output, can add files as input for execution
  <# (coloringModule) in = arcs::arc,nodes::node; out = res::col; FILE="./f
    col(X,"red") v col(X, "green") v col(X, "blue") :- node(X).
    :- col(X,C), col(Y,C), arc(X,Y).
  #>
  if_no_answerset {
      System.out.println("No solution");
      return null;
  }
  catch_error{
      System.out.println("Error");
      return null;
  }
```

No consoles to display at this time.

Overview

JDLV plug-in for the Eclipse IDE is designed to give you a powerful, integrated environment for building applications based on JASP specification. JASP is a hybrid language that transparently supports a bilateral interaction between ASP (Answer Set Programming) and Java.

Requirements

A compatible version of Eclipse is required in order to install or use the JDLV plug-in. Currently, the available plug-in has been tested on 3.6 and 4.2 Eclipse versions.
It is strongly recommended to install the plug-in to version 4.2 (Juno) to be sure that the plug-in works correctly. You can download Eclipse at https://www.eclipse.org.

JDLV plug-in installation

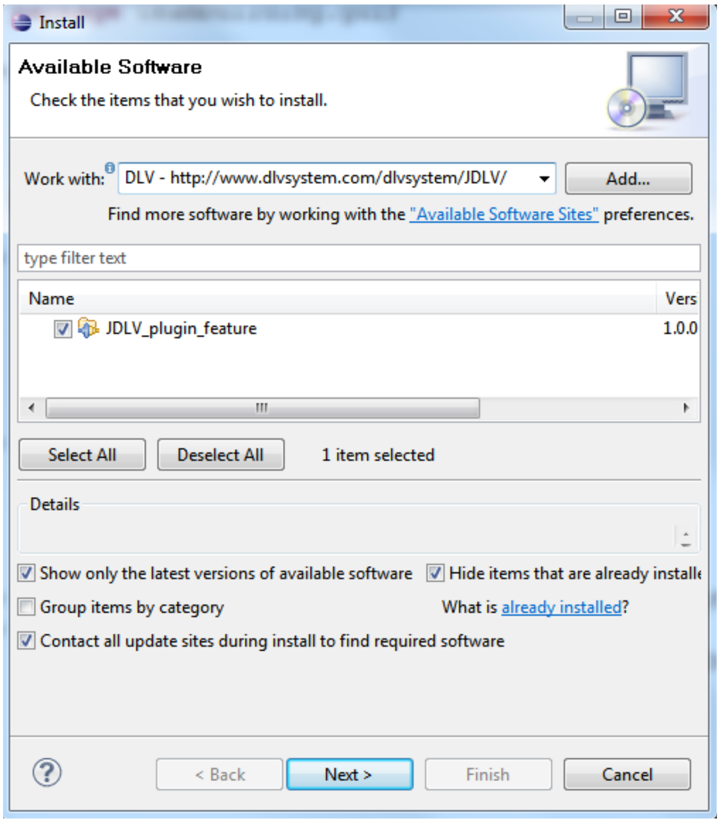
To install JDLV follow steps below:

Start Eclipse, then select **Help** > **Install New Software...**.

1. Click **Add**, in the top-right corner.
2. In the Add Repository dialog that appears, enter "JDLV Plugin" as *Name* and the following URL as *Location*: http://www.dlvsystem.com/JDLV_plugin/
3. Click **OK.**
4. **In the case where the Add Repository dialog does not show the *JDLV_plugin_feature,* please remove the selection at Group item by Category.**
5. In the **Available Software dialog**, select the checkbox JDLV, enable the possibility to **contact the available update sites** during the installation and click **Next**.

   NOTE: this operation of calculating required software for installation may take a few minutes.

6. In the next window, you will see a list of the tools to be downloaded. Click **Next**.
7. Read and accept the license agreements, then click **Finish**.
8. During the installation, a warning message about unsigned libraries may appear.
9. When the installation is finished, please restart Eclipse.



Install

Available Software

Check the items that you wish to install.

Work with: DLV - http://www.dlvsystem.com/dlvsystem/JDLV/

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Vers
☑ JDLV_plugin_feature	1.0.0

1 item selected

Details

☑ Show only the latest versions of available software ☑ Hide items that are already installe

☐ Group items by category

What is already installed?

☑ Contact all update sites during install to find required software

Using the plug-in

JDLV plug-in works in a standard **Java Project:** if a project is available in the Eclipse workspace, you have to add the **Xtext Nature** to enable the JDLV functionalities.

- Right click on the project node in the Project Explorer Tree > **Configure > Add Xtext Nature.**

Now the project is ready to manage JDLV files. In order to add a new JDLV file follow

1. Create new file with .jdlv extension

   Note: the creation of a new JDLV file creates automatically, in the same position, a new Java file with the same name.

2. Open the just created .jdlv file with the JDLV editor and write your first JASP code!

3. After saving your JASP code, the system automatically compiles it, storing the result in the corresponding Java file.

Project configuration

The Java file compiled (see previous section) need external java Libraries for the execution: just set the **build path** of project by adding the following libraries

- **JDLVExecutor.jar**
- **DLVWrapper.jar**.

DLVWrapper.jar can be downloaded at http://www.dlvsystem.com/dlvwrapper.

DLV code need an external solver for its execution (you can download the right DLV solver version at http://www.dlvsystem.com/DLV)
To specify a solver for your project, you can create in the root of the project a **java file properties** with name ***jdlv.properties***. Here you can specify your executable as follows:
executable_dlv=.\solver_path
In ***jdlv.properties*** file you may want to disable log messages that appear during program execution by setting the *log_level* property as log_level=NONE.

JDLV language by example

JDLV language supports different TAG to create a DLV program embedded in a Java procedure.

```
The tag        <#+ (JDLV_program)
                    “JDLV_head”
                    “DLV CODE”
               #>
```

allows to append DLV code, mappings, options in the specified program (in this example called “JDLV_program”). A JDLV file, can manage more than one JDLV program, and the various programs will be distinguished according to their name (note that it will not start the execution).

```
The tag        <# (JDLV_program)
                    “JDLV_head”
                    “DLV CODE”
               #>.( “option of execution” )        “JDLV Handlers”
```

allows to start the execution of specified program and manage the result in the JDLV handlers.
In DLV option section you can use all options available for the solver.

JDLV head

In this section you can specify mappings, import of external DLV files and the solver path to use for the execution:

- **IN** – allows to import Java variables (usually Collections) in the program. The objects content in the collection are translated in DLV code that will be used at execution time.

  Example: List<Arc> arcs; **<#+(program) IN**=arcs::arc; **#>** appends code “generated” from objects content in the *arcs* collections using *arc* as predicate name. If the predicate name is omitted, JDLV uses the variable name as predicate name.

  If the predicate name is “@” JDLV uses the class name as predicate name.

- **OUT** – allows to export in Java variables (usually Collections) the output predicates of the execution. All facts mapped as output are transformed in java objects and stored in the mapped Java variables.

  Example: List<Colored> color; **<#+(program) OUT**=colored::col; **#>** transforms the *col* predicates (result of the execution) in *Colored* objects and stores them in *color* collection.

  If the predicate name is omitted, JDLV uses the variable name as predicate name.

  If the predicate name is “@” JDLV uses the class name as predicate name.

- **INOUT -** allows you to use variables for the input and for the output of the program.

- **FILE –** allows to import external file as input for the execution

  Example: <#+(program) FILE=”file_path”; **#>**

- **EXE –** allows to specify the solver to be used for the execution

  Example: <#(program) EXE=”solver_path”; **#>**

JDLV handlers

JDLV handlers are specified to manage the output of JDLV execution. There are three different handlers:

- **foreach_answer_set –** the Java code specified in this block is executed for each answer set computed from solver. For each model, the content of variables specified as output (using out or inout mapping) is updated with the values of objects obtained from the new model.

  Example: <#(program)#>for_each_answerset{print(outputVariable);}

- **if_no_answer_set –** the Java code specified in this block is executed when the execution of JDLV code does not admit any solution.

  Example: <#(program)#>if_no_answerset{print(“No solution”);}

- **catch_error –** the Java code specified in this block is executed when the execution of JDLV generates an exception.

  Example: <#(program)#>catch_error{print(“ERROR!!”);}

3Colorability example code

In this example we use the class *Arc* to represent a graph. An instance of graph is used in JDLV to resolve the 3Colorability problem.
The classes used in JDLV as input or as output must have getter and setter for each field.
A class is translated in DLV fact by using fields in order of declaration.

An object of Arc class is automatically transformed in the follow DLV fact:
`Arc(node1, node2).`

File Arc.java

```
public class Arc {
  String node1;
  String node2;

  public Arc(String start, String end) {    node1=start; node2=end;   }

  public String getNode1() {    return node1;   }

  public void setNode1(String node1) {    this.node1 = node1;   }

  public String getNode2() {  return node2;   }

  public void setNode2(String node2) {   this.node2 = node2;   }
}
```

The class Colored is mapped for the result of JDLV execution: it represents the color assigned to a node.

File Colored.java

```
public class Colored {
  public String node;
  public String nodeColor;

  public String getNode() {     return node;  }

  public void setNode(String node) {    this.node = node;  }

  public String getNodeColor() {    return nodeColor;  }

  public void setNodeColor(String nodeColor) {    this.nodeColor = nodeColor;  }

  public String toString() {    return node+"->"+nodeColor;  }
}
```

The JDLV file is a Simple Java class but in the *computed3Coloring()* method it uses JDLV to compute the 3Colorability problem.

File Graph.jdlv

```
public class Graph {
  private List<Arc>arcs=new ArrayList<Arc>();//Arcs of graph are instances of
Arc class
  private List<String> nodes =new ArrayList<String>();//Nodes are simple String

  public Graph(){
    //create a graph composed from only 5 nodes
    //other nodes will be added in compute3Coloring() method
    for(int i=1; i<=5; i++){
      addNode("n"+i);
    }

    //add arcs objects in this graph
    addArc(new Arc("n2", "n4"));
    addArc(new Arc("n2", "n3"));
    addArc(new Arc("n3", "n5"));
    addArc(new Arc("n4", "n6"));
    addArc(new Arc("n4", "n5"));
    addArc(new Arc("n5", "n7"));
    addArc(new Arc("n6", "n8"));
    addArc(new Arc("n6", "n7"));
    addArc(new Arc("n7", "n9"));
    addArc(new Arc("n8", "n10"));
    addArc(new Arc("n8", "n9"));
    addArc(new Arc("n9", "n1"));
    addArc(new Arc("n10", "n1"));
  }

  public List<Colored> compute3Coloring() {
    List<Colored> res = new ArrayList<Colored>();

    //to compute 3Coloring use a JDLV module called coloringModule
    //in a JDLV module can add simple DLV code. In this portion add two facts of
node
```

```
    <#+  (coloringModule)
      node("n6").
      node("n7").
    #>

    //can add DLV code using strings obtained evaluating Java expressions
    for(int i=8; i<=10; i++){
      <#+ (coloringModule)
              node(${"n"+i}).
      #>
    }

    //add mapping for input and output,
    //can add files as input for execution
    <# (coloringModule)
      IN = arcs::arc;//add objects content in arcs collection using arc
predicate name
      IN = nodes::node;//add strings content in nodes using node predicate name
      OUT = res::col;//use res collection to store object created from col
predicate
      FILE="./files/arcsInstances.dl";//append DLV code content in the selected
file

      //simple DLV code used to compute 3Coloring problem
      col(X,"red") v col(X, "green") v col(X, "blue") :- node(X).
      :- col(X,C), col(Y,C), arc(X,Y).
    #>//when JDLV call module (<# #>) is closed
      //the execution of JDLV program start automatically
    .("-n=1")//can add option of execution separated by ‘,’
    if_no_answerset {
      //this portion of Java code is executed if DLV execution does not admit
solution
      System.out.println("No solution");
      return null;
    }
    catch_error{
      //this portion of Java code is executed if JDLV execution generates an
exception
      System.out.println("Error");
      return null;
    }
    //if JDLV execution have a solution, res collection is automatically updated
    return res;
  }

  public void addNode(String id) {    nodes.add(id);   }

  public void addArc(String from, String to) {    arcs.add(new
Arc(from,to));   }

  public void addArc(Arc arc) {    arcs.add(arc);   }

  public List<Arc> getArcs() {    return arcs;  }

  public void setArcs(List<Arc> arcs) {    this.arcs = arcs; }

  public List<String> getNodes() {    return nodes;  }
  public void setNodes(List<String> nodes) {    this.nodes = nodes;  }

}
```