Witmate IDE Users Manual

Version 2006.3

mate of your wit

Contents

Introduction

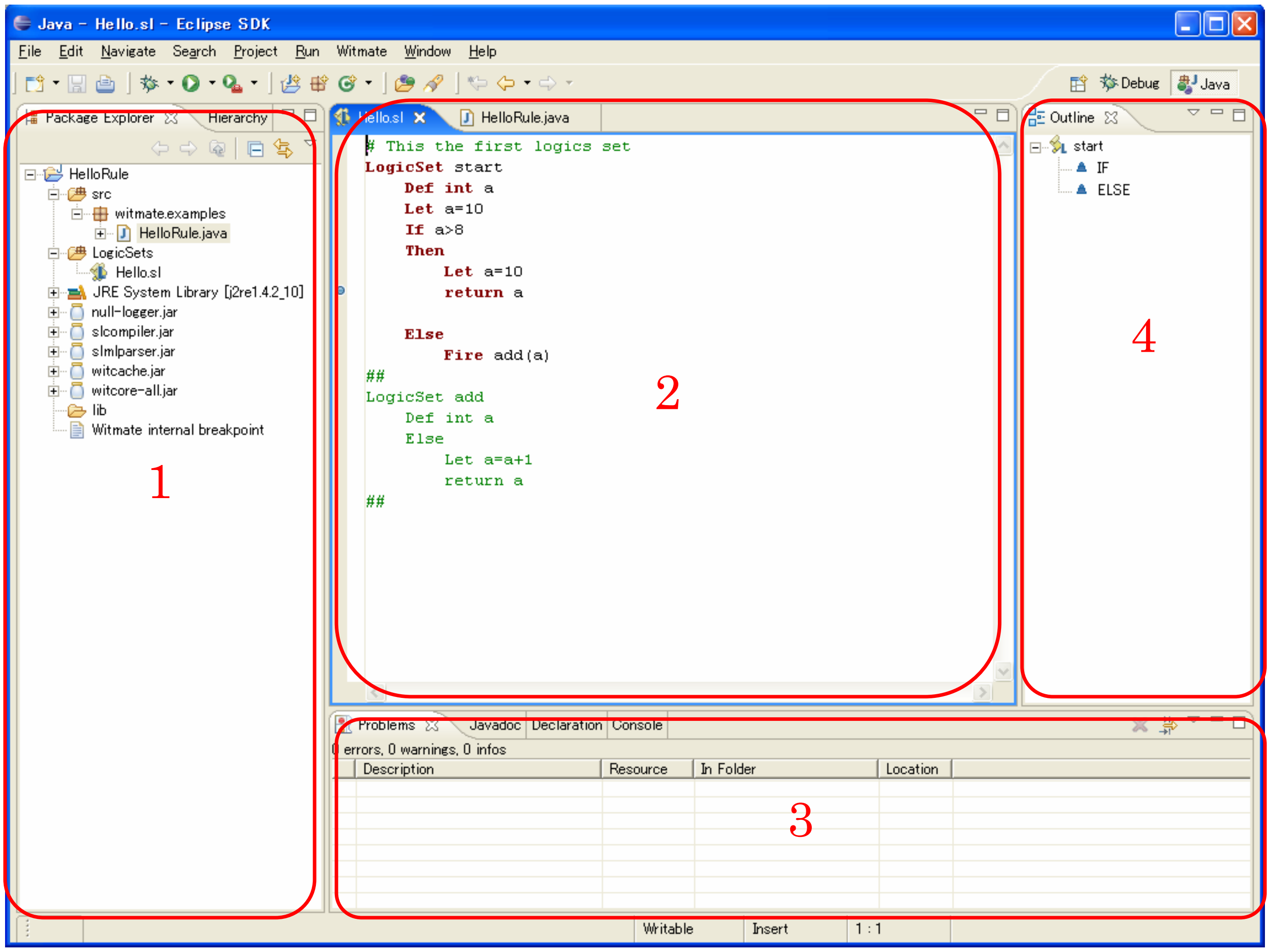
Witmate IDE is the Integrated Development Environment to develop/debug logics written in Witmate Simple Logic language for logics developers and managers.
Witmate IDE is a commercial extended edition of Eclipse the open source IDE platform. So it's very familiar and easy environment for users who have experiences with Eclipse. It's recommended to read manuals of Eclipse to users without Eclipse experiences before reading this manual. Eclipse manuals are on http://www.eclipse.org/documentation/main.html.
The functionalities of WitmateIDE is separated into 3 features sets, or called Perspectives in Eclipse words, that are Features of Witmate Simple Logic language creation, debugging and menu. All these features are explained detailed in this manual. After that, a simple tutorial is introduced.

Features

Features of Witmate Simple Logic language creation

These features construct environment to create logics by Witmate Simple Logic language. It is a extension of Java Perspective of Eclipse, so users need not to switch Perspectives between Java and Simple Logic language creation.

WitmateIDE window is typically separated into 4 view areas as below.



```
# This the first logics set
LogicSet start
    Def int a
    Let a=10
    If a>8
    Then
        Let a=10
        return a

    Else
        Fire add(a)
##
LogicSet add
    Def int a
    Else
        Let a=a+1
        return a
##
```

0 errors, 0 warnings, 0 infos

1. Navigator of project

   List all elements of the projects which are opened now.

2. Source code view

   Show/Edit source codes that are opened. Tabbed view is supported.

3. Output

   Show system output of creation or running as problems of source code, console output etc.

4. Outline

   Show outline of source code that is on the top of tabbed view.

Details about these areas are explained below one by one.

- Navigator of project

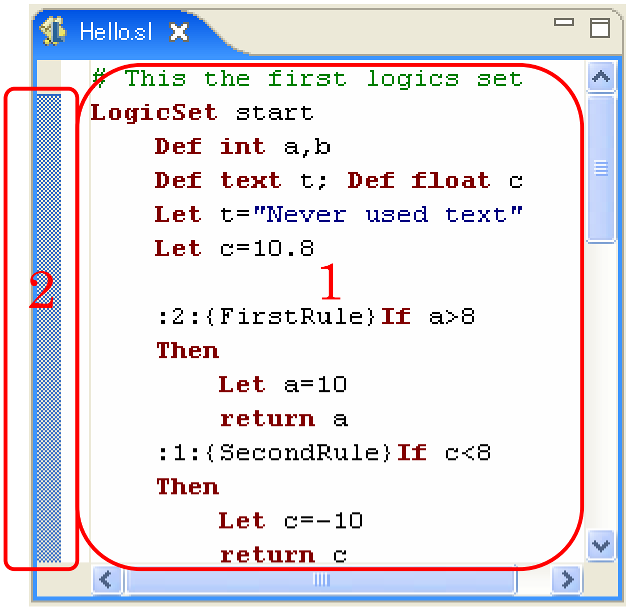
This is the navigator of Java project with a Witmate simple logic language files extension. All simple logic language files are shown with a SL icon.

Double click the name of SL file, that file will be opened in source code view.

- Source code view

All source code files that are opened are shown in this tabbed view to be ready to be edited.

Any SL file that is opened will be shown in one SL editor as one tab in this view and shown a SL icon in its tab. This is a sample SL editor:



```
# This the first logics set
LogicSet start
    Def int a,b
    Def text t; Def float c
    Let t="Never used text"
    Let c=10.8

    :2:{FirstRule}If a>8
    Then
        Let a=10
        return a
    :1:{SecondRule}If c<8
    Then
        Let c=-10
        return c
```

1. The main part of SL editor for showing and editing source
2. This is an area named Marker bar to show and expose special functions for SL language file. Details of the Marker bar are explained feature details below.

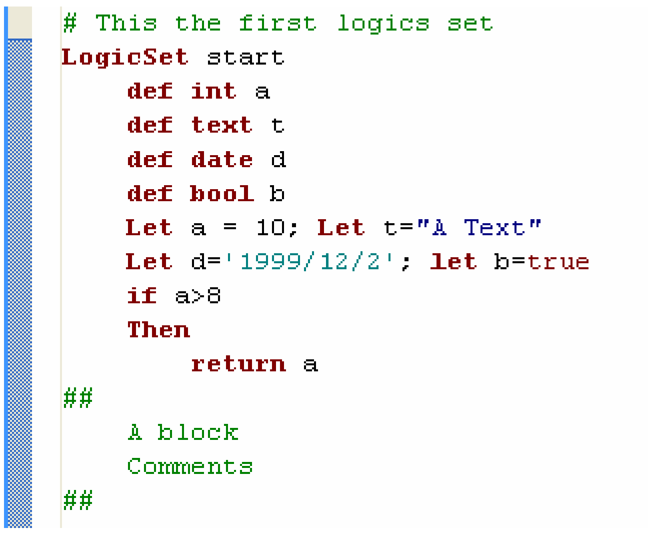
There are assistant functions in SL editor to make edit SL language easier.

➢ Colored contents

Some of items of SL are shown in special colors to highlight them.

Item type	Color	Style
Key words	Dark red	Bold
Text constants	Blue	Normal
Date constants	Dark green	Normal
Boolean constants	Red	Normal
Comments	Light green	Normal

Below is an example of colored contents.

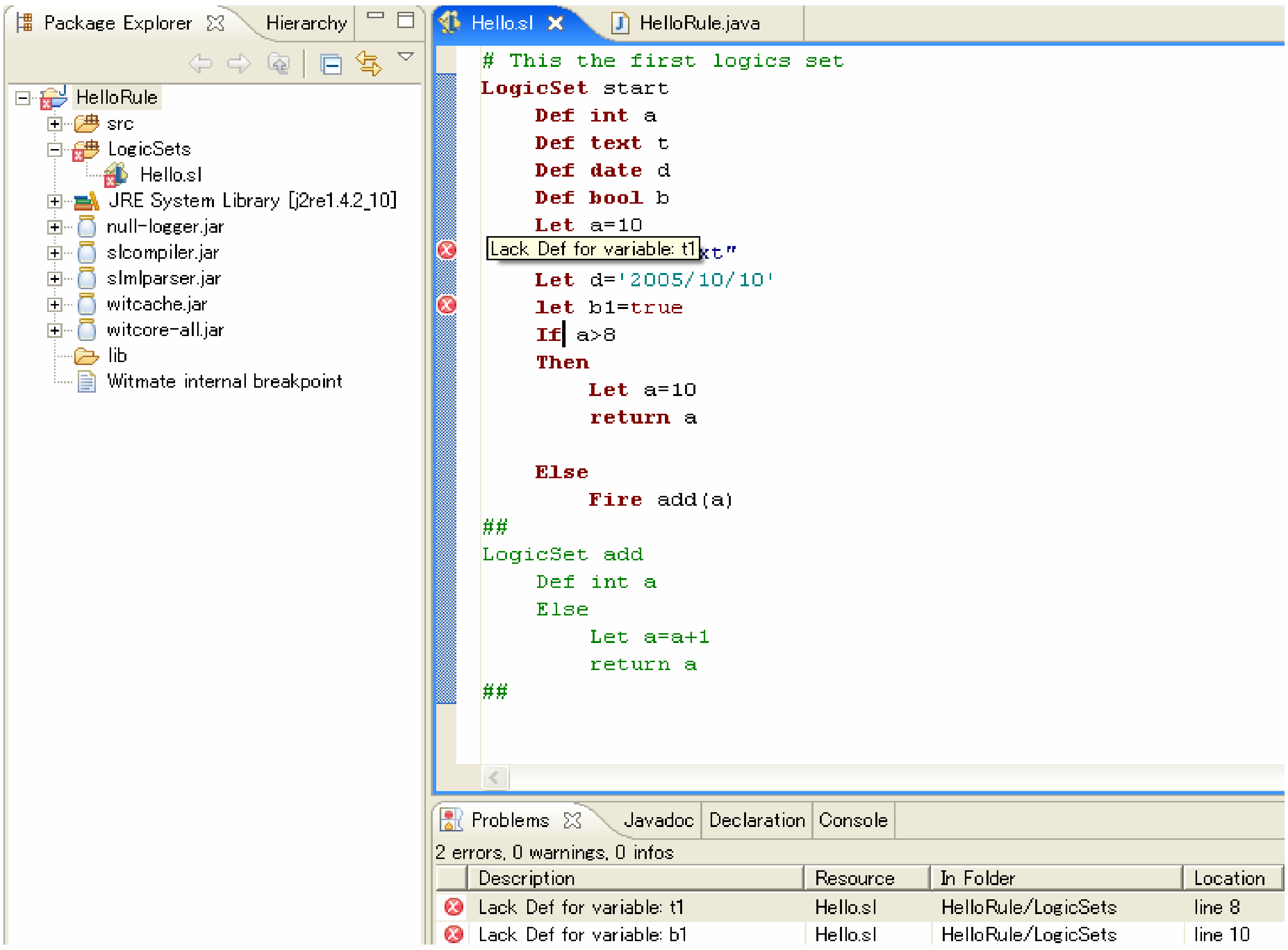


```
# This the first logics set
LogicSet start
    def int a
    def text t
    def date d
    def bool b
    Let a = 10; Let t="A Text"
    Let d='1999/12/2'; let b=true
    if a>8
    Then
        return a
##
    A block
    Comments
##
```

- Error marking

At the same time of editing a SL file, Witmate IDE checks SL and figure out any errors with an error mark in left side of SL editor. And these errors are listed in Problem tab in Output area too. There are 2 ways to check the details of errors. The first one is moving your mouse cursor on these error marks, the details of errors will be shown in a popup window. The second one is check them in Problem tab of Output view. Double clicking the row of certain error in Problem tab will highlight the line which made this error.

An error mark will be shown upon the SL icon of certain SL file which has any errors in Navigator of project view.



```
# This the first logics set
LogicSet start
    Def int a
    Def text t
    Def date d
    Def bool b
    Let a=10
Lack Def for variable: t1 xt"
    Let d='2005/10/10'
    let b1=true
    If a>8
    Then
        Let a=10
        return a

    Else
        Fire add(a)
##
LogicSet add
    Def int a
    Else
        Let a=a+1
        return a
##
```

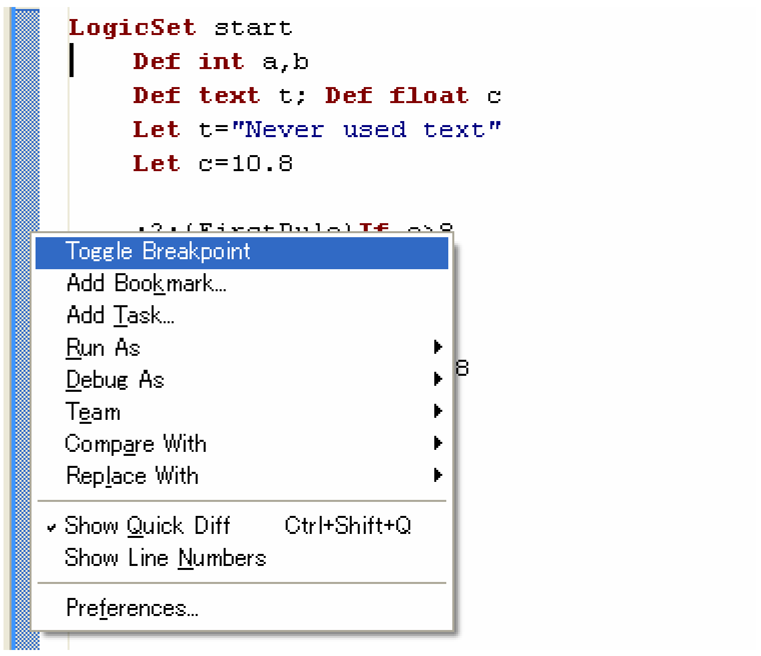
2 errors, 0 warnings, 0 infos

	Description	Resource	In Folder	Location
●	Lack Def for variable: t1	Hello.sl	HelloRule/LogicSets	line 8
●	Lack Def for variable: b1	Hello.sl	HelloRule/LogicSets	line 10

- Toggle breakpoints

There are to ways to toggle a breakpoint in SL file.

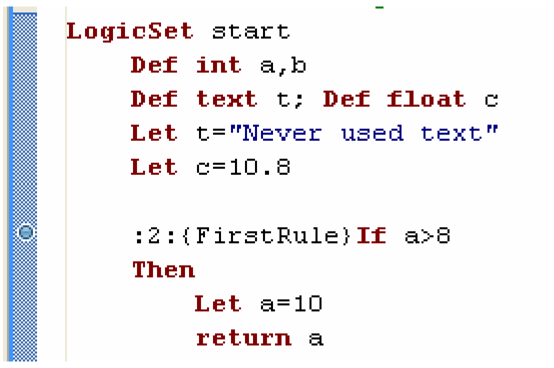
1. Right click the Marker bar before the line at which the breakpoint will be set, and click the Toggle Breakpoint item in the popup menu as the example:



```
LogicSet start
    Def int a,b
    Def text t; Def float c
    Let t="Never used text"
    Let c=10.8
```

2. Double click the Marker bar before the line at which the breakpoint will be set.

A blue ball is shown in Marker bar as a sign of breakpoint as below shown.



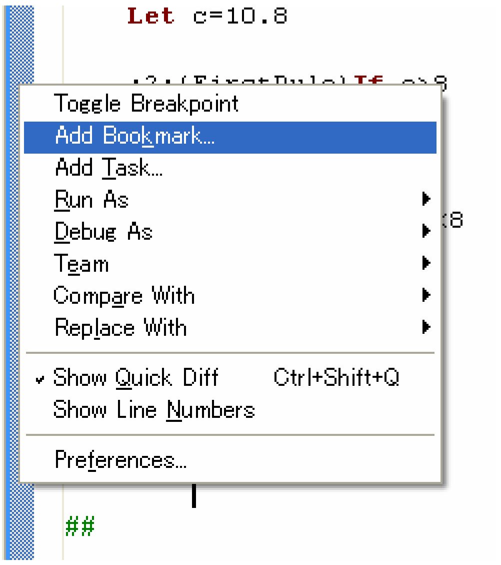
```
LogicSet start
    Def int a,b
    Def text t; Def float c
    Let t="Never used text"
    Let c=10.8

    :2:{FirstRule}If a>8
    Then
        Let a=10
        return a
```

Toggling the breakpoint again removes this breakpoint.

! Not every line of SL language can be set breakpoints, so if toggle a breakpoint at a line that can not be set breakpoint, nothing will be shown on Marker bar.

➢ Other general functions of Eclipse source editors

Because SL editor is an extension of Eclipse source editor, most of general functions exposed by Eclipse are used in SL editor e.g. Set Bookmarks:



Refer Eclipse manual for usages of other functions.

● Output

As explained above, there are several tabbed views in Output area as Problem, Console and more.

- Outline

The structure of Logic sets in one SL file is shown in Outline.

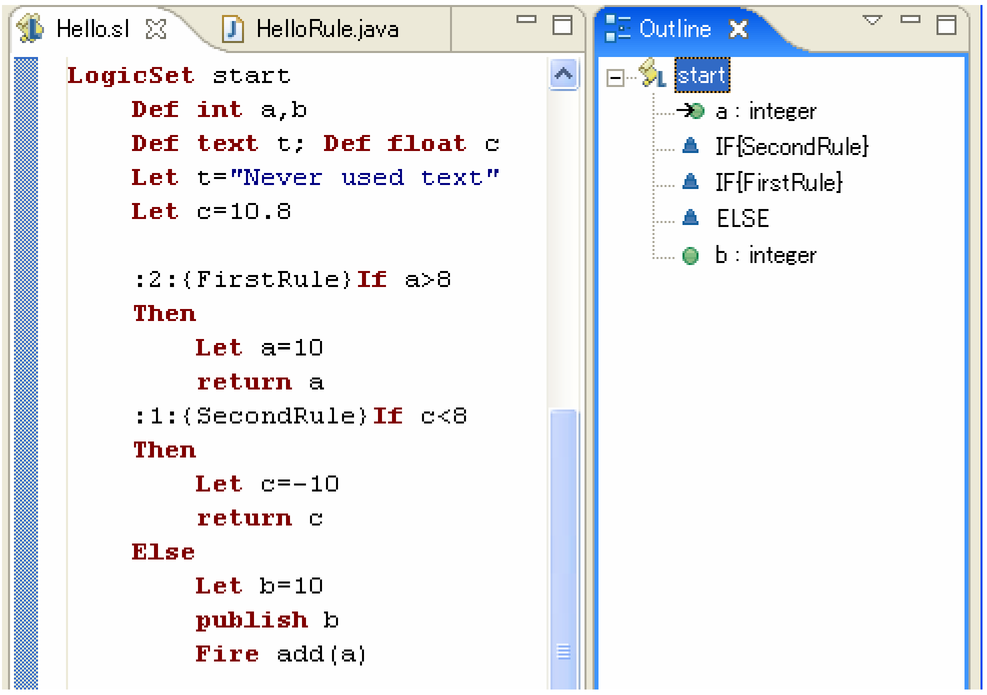
Elements below of logic set are shown in outline:

- Input variables with icon
- IF…THEN with icon , name of rule and sort by Sort number of rule
- Else with icon
- Published variables with icon

! Types of Variables below are not shown in outline:

i) Never used in any rules as the variable t in the example below

ii) Used only in logic set as the variable c in example below

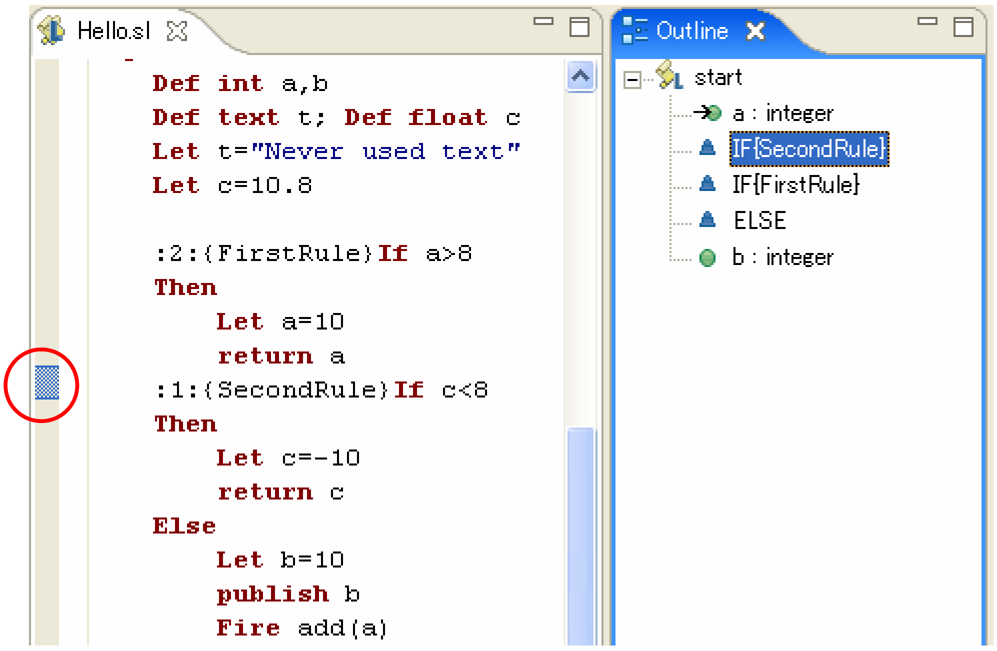


```
LogicSet start
    Def int a,b
    Def text t; Def float c
    Let t="Never used text"
    Let c=10.8

    :2:{FirstRule}If a>8
    Then
        Let a=10
        return a
    :1:{SecondRule}If c<8
    Then
        Let c=-10
        return c
    Else
        Let b=10
        publish b
        Fire add(a)
```

```
start
    a : integer
    IF{SecondRule}
    IF{FirstRule}
    ELSE
    b : integer
```

Clicking the logic set name, IF or Else elements in outline, scrolls and highlights the element which is clicked in source code view. The example above is the result when clicked the logic set name “start”.

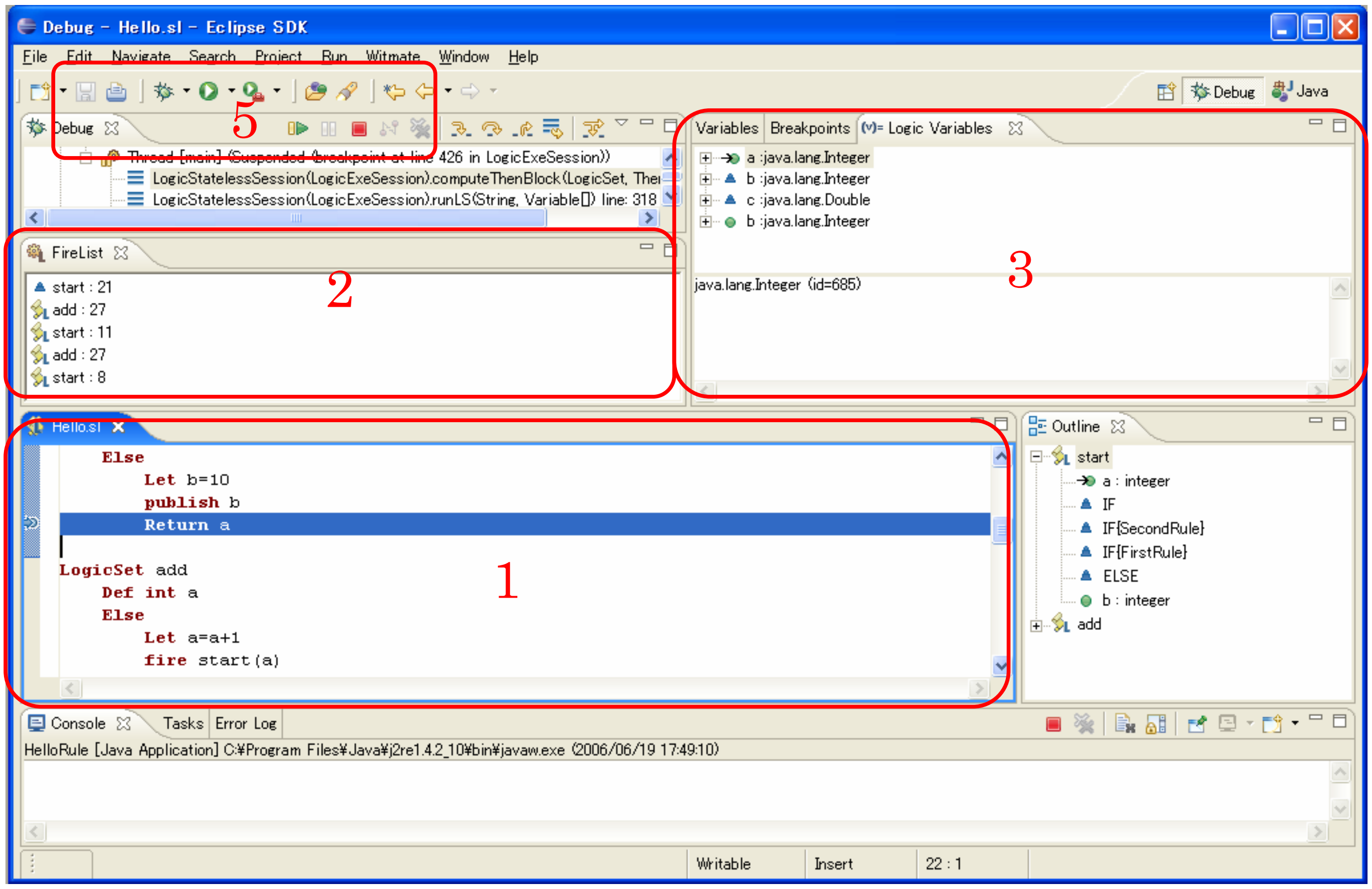
The example below is shown the result when clicked the element "IF{SecondRule}". Pay attention on the Marker bar on left of source code view that figures out the location of the rule clicked in source.



```
Def int a,b
Def text t; Def float c
Let t="Never used text"
Let c=10.8

:2:{FirstRule}If a>8
Then
    Let a=10
    return a
:1:{SecondRule}If c<8
Then
    Let c=-10
    return c
Else
    Let b=10
    publish b
    Fire add(a)
```

Features of Witmate Simple Logic language debugging

To troubleshoot any mistakes or bugs in SL language, SL debugger is exposed. SL debugger is an extension of Java debugger, a typical view of SL debugger is as:


Debug - Hello.sl - Eclipse SDK
File Edit Navigate Search Project Run Witmate Window Help
Debug
Java
Debug
Variables Breakpoints Logic Variables
FireList
Hello.sl
Outline
Console Tasks Error Log
Writable
Insert
22 : 1
5
2
3
1

```
Thread [main] (Suspended (breakpoint at line 426 in LogicExeSession))
    LogicStatelessSession(LogicExeSession).computeThenBlock(LogicSet, Ther
    LogicStatelessSession(LogicExeSession).runLS(String, Variable[]) line: 318
```

```
start : 21
add : 27
start : 11
add : 27
start : 8
```

```
a :java.lang.Integer
b :java.lang.Integer
c :java.lang.Double
b :java.lang.Integer

java.lang.Integer (id=685)
```

```
    Else
        Let b=10
        publish b
        Return a

LogicSet add
    Def int a
    Else
        Let a=a+1
        fire start(a)
```

```
start
    a : integer
    IF
    IF{SecondRule}
    IF{FirstRule}
    ELSE
    b : integer
add
```

```
HelloRule [Java Application] C:¥Program Files¥Java¥j2re1.4.2_10¥bin¥javaw.exe (2006/06/19 17:49:10)
```

Besides of those views which are same as SL creation perspective and general Java debugger perspective, these views are extended for SL debug:

1. Source code view with Instruction Pointer

   This view has all functions same as source code view of SL creation, plus showing an Instruction Pointer ➧ on the marker bar to figure out a stop at a breakpoint.

2. Fire list view

   A list of fire commands locations in source code, through which the rule match stopped at current breakpoint. The fire command is farer from current location is lower in this list. Current breakpoint stop is at the top of this list with icon ▲ . Double-clicking a line of this list rolls the source code to this location and highlights that line. And an Instruction Pointer ➧ is shown n the marker bar to figure out that line at the same time.

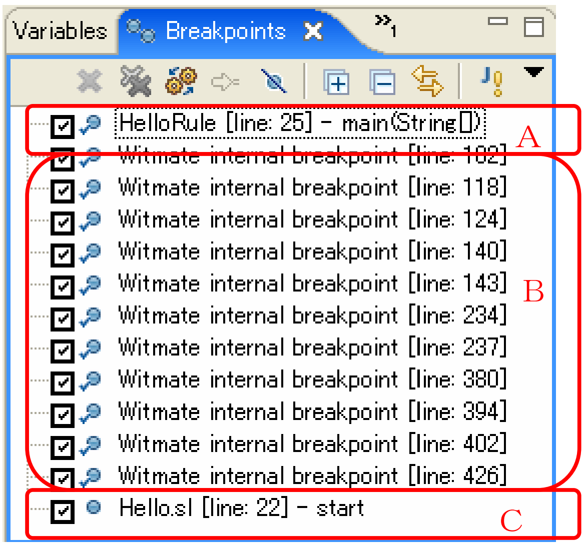
   ! This is not stack.

3. Logic Variables view

   All variables that are in the scope of the current stop of breakpoint are dumped in this view. Because the fire list view is not stack, even clicked fire list item, this variables view is still keep no change.

4. Breakpoint view

   Breakpoint view is normally tabbed with logic variables view in a same area, so it is not visible in the above example. Below is an example of breakpoint view when IDE stopped at certain breakpoint:



```
HelloRule [line: 25] - main(String[])
Witmate internal breakpoint [line: 102]
Witmate internal breakpoint [line: 118]
Witmate internal breakpoint [line: 124]
Witmate internal breakpoint [line: 140]
Witmate internal breakpoint [line: 143]
Witmate internal breakpoint [line: 234]
Witmate internal breakpoint [line: 237]
Witmate internal breakpoint [line: 380]
Witmate internal breakpoint [line: 394]
Witmate internal breakpoint [line: 402]
Witmate internal breakpoint [line: 426]
Hello.sl [line: 22] - start
```

   This view is an extension of general Java breakpoint view, so there may be:

   A. General Java breakpoints, clicking these breakpoints makes source code view to show the Java source code including this breakpoint and highlight the location of the breakpoint clicked.
   B. Witmate internal breakpoints: Managing by Witmate IDE, do not remove or disable these breakpoints
   C. SL breakpoints: format is File name [line: line number] – LogicSet name, clicking these breakpoints makes source code view to show the SL source code including this breakpoint and highlight the location of the breakpoint clicked.

5. Debug Actions

   These are action buttons to boot/continue debugging.

   Normally, boot debug by click Debug button, or right click the java class that will be debugged and select certain debug action.

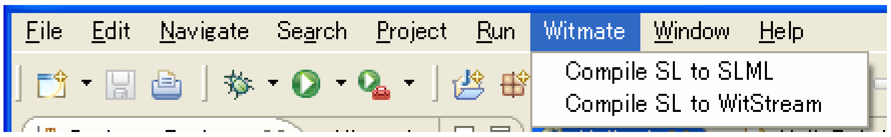
   When IDE stopped at one SL breakpoint, clicking Resume button continue debug to end or another breakpoint.

   ! SL debugger does not support step debug, so do not try to use step in/out buttons, just use Resume.

   Clicking Terminate button stop the debugging.

Features on menu

There is a Witmate menu with 2 actions in the menu bar as below shown:



When a SL file in source code view is on the top of the tabs, these actions are enabled.
As the menu item expressed, they compile the top SL file and output a SLML file or WitStream file with the same name of SL file following file name extension slml or wits.
! These menu actions will rewrite any files with same name, so backup files created to a safe place before using these menu actions.

Typical scenarios to use Witmate IDE

1. Use the example Java project or create your own Java/Web project.
2. Write the engine loading codes referring the example project or SDK users' manual
3. Write your SL files helping by Features of Witmate Simple Logic language creation
4. Run the engine loading codes
5. If these is any mistakes, set breakpoints at locations you concern, and Debug the engine loading codes
6. Check the Fire list and Logic variable view to trouble shoot your SL logics, and stop the debugger, fix these bugs
7. Repeat step 5 and 6 until all mistakes gone
8. (Optional) Using menu action Witmate→ Compile SL to SLML to generate SLML file, then copy this SLML file to any place of your system to be ready to exchange these logics with other systems.
9. (Optional) Using menu action Witmate→ Compile SL to WitStream to generate WitStream file, then copy this SLML file or the contents of this file to your code, database SQL or any place of your system to be ready to store or to broadcast these logics .
10. Deploy Logis with Witmate required components into execution environment, and use it!

How to

This chapter explains the solutions for special situations.

Projects

How to create your own Java projects

Follow these steps to create your own project from beginning.

1. Create java project or Web project
2. Create a SL file folder to contain SL files
3. Include the SL file folder into project source paths using Project➔Properties dialog.

SL creation

How to move cursor to a certain location of SL

Click logic set name in outline to move the cursor to that logic set, or click IF/ELSE item in certain logic set outline to move the cursor to the location of the IF/ELSE item.

SL Debugging

How to move cursor to a certain fire location

Click one row of Fire list to move the cursor to that location the fire command executed.

Performance tuning

How to save the SL compile time for debugging

If you are debugging a very long SL file or many SL files, it will take long time to compile these SL before every time you can debug them. To save these compile time, use Witmate→ Compile SL to WitStream menu action to compile these SL into WitStream and change your engine loading codes to load these WitStream files instead of loading SL file themselves.

Troubleshooting

Cursor can not stop at correct line of breakpoints in Logic set sources

It may be caused by 2 reasons:

1. Forget to include logic sets source folder in project source path

To stop at breakpoints in Logic set sources, Logic set sources have to be included in source path of project.

2. There are logics which have same logic set name