RhapsodyGuru

Customizing Rhapsody - Writing Helpers

Contents

Overview: Rhapsody Helper Applications

Helpers in Rhapsody are programs, that you attach to IBM® Rational® Rhapsody® to extend its functionality. They can be either external program or in Java written Application that use the Rational Rhapsody API. Helpers may be invoked manually from the Tools menu, or automatically, based on some event, for example 'before saving'.

These Java written Application that use the Rhapsody API can be created as:

Standalone applications	When invoked the helper loads, runs as an external program and when it has done its job it terminates.
Plug-ins	Loads when the Rhapsody project loads. When invoked, it runs as part of the Rhapsody process. Only unloads when Rhapsody terminates.

In this short abstract we're going to the necessary steps to write a little helper, which can be invoked from the Rhapsody Browser. Our helper will extend Rhapsody functionality by locating the current the *Active Component* in the browser hierarchy. This missing Rhapsody feature is helpful, especially when you have to deal with bigger projects and nested browser hierarchy.

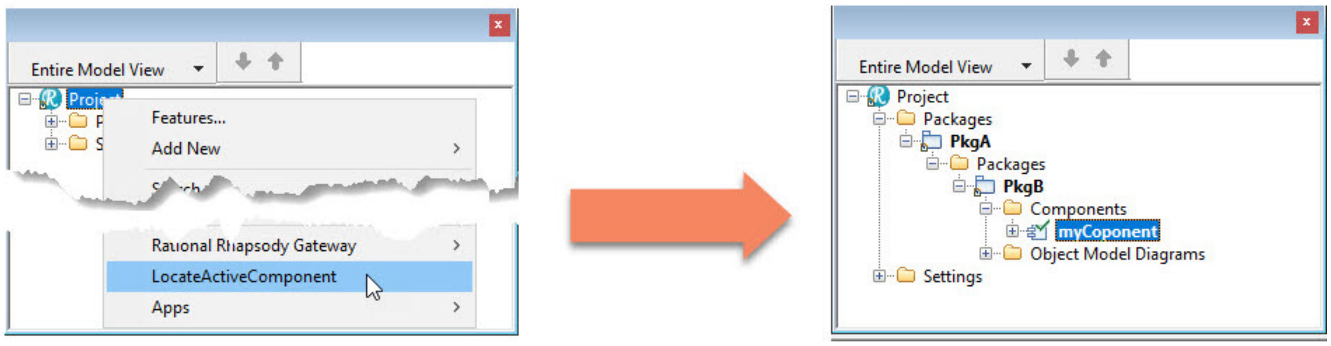


Figure 1: Locate Active Component in Browser

Task 1: Writing a Helper Application

Extending Rhapsody functionality by writing your own application is no rocket sine: A running Rhapsody instance, Eclipse and Java SDK Environment are the initial ingredients that you need to get started.

Figure 2: Three ingredients to customise and extend Rhapsody

Make sure that you have the latest Eclipse release installed on your machine, To avoid any issues make sure that the Rhapsody and Eclipse Bit-Version is consistent. Mixing the "Bit-Version" might result in unintended and unexpected behaviour. You can comibine>

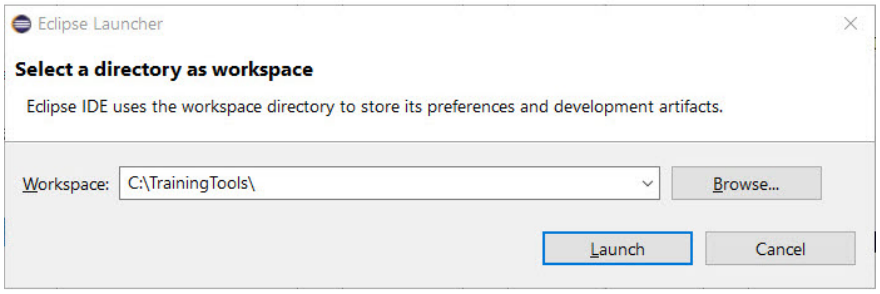
- Rhapsody-32Bit with Eclipse-32Bit or
- Rhapsody-64Bit with Eclipse-64Bit

In this Task we will create an Eclipse Project and using the Rhapsody API to write our own Helper Application

1. Launch Eclipse and create project:

a) Launch Eclipse

b) Chose `c:\TrainingTools\` as WorkSpace



Eclipse Launcher

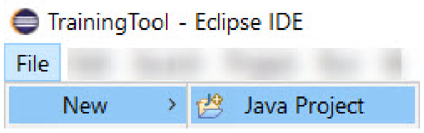
Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\TrainingTools\

Figure 3: Eclipse Launcher

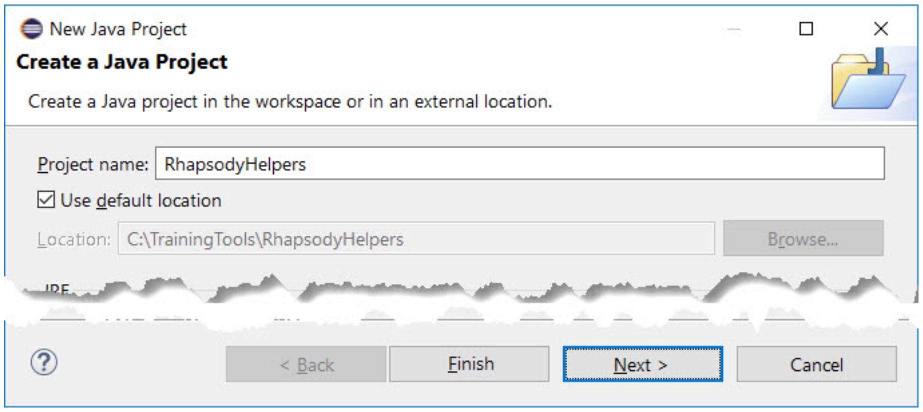
c) From the Eclipse menu select **File > New > Java Project**



TrainingTool - Eclipse IDE

Figure 4: New Java Eclipse project

d) Name the project `RhapsodyHelpers`



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: RhapsodyHelpers

☑ Use default location

Location: C:\TrainingTools\RhapsodyHelpers

Figure 5: Project Name: RhapsodyHelpers

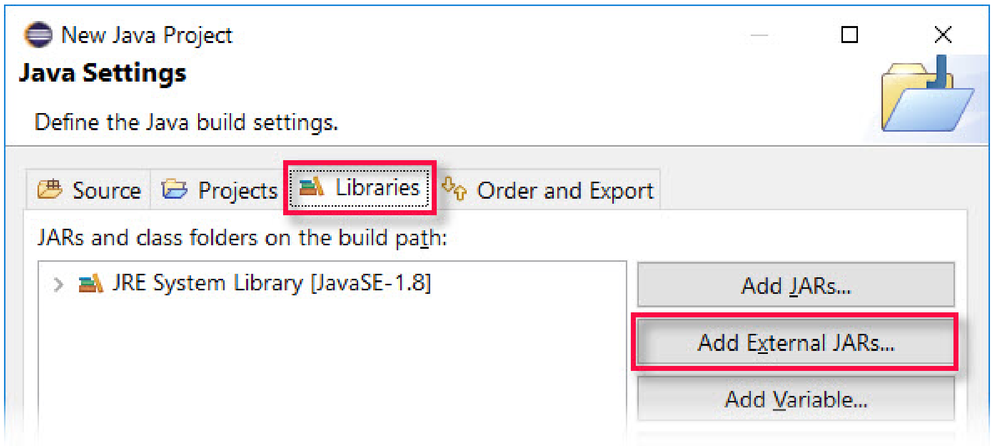
e) Click **Next>**

2. Java Settings - Referencing the Rhapsody Java API:

For each new project in Eclipse you must specify in the **Java Build Path:**

- The location of the *Rhapsody API JAR* (Rhapsody.jar)
- The location of the *Rhapsody Native Library* (Rhapsody.dll)

a) In **Java Settings** Window select the **Libraries** tab



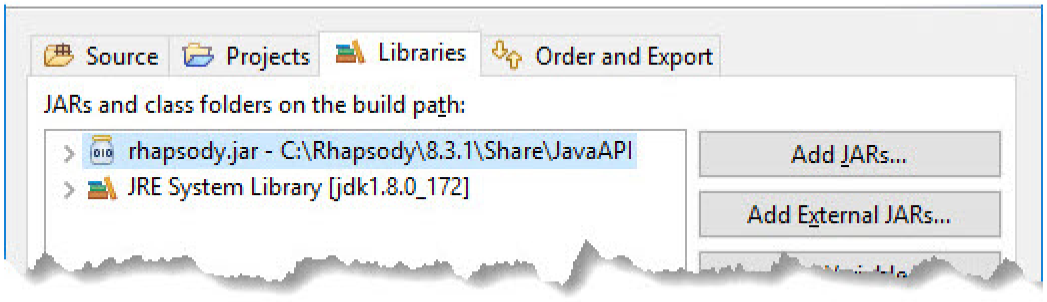
JARs and class folders on the build path:

JRE System Library [JavaSE-1.8]

Figure 6: Project Configuration - Libraries Tab

b) Click **Add External JARs...**

c) Select `rhapsody.jar` from `\Share\JavaApi` directory.



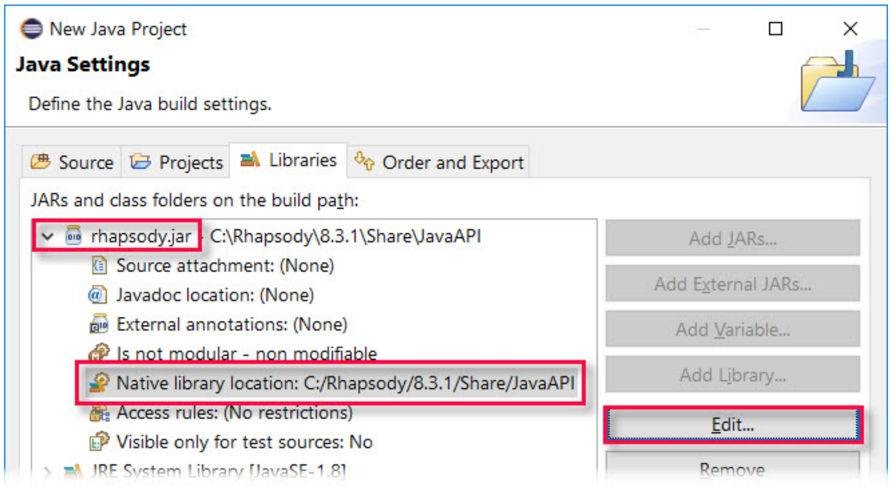
JARs and class folders on the build path:

rhapsody.jar - C:\Rhapsody\8.3.1\Share\JavaAPI

JRE System Library [jdk1.8.0_172]

Figure 7: Project Configuration - Adding rhapsody.jar in build path

d) Expand `rhapsody.jar` and edit the **Native library location** to point `[...]\Share\JavaAPI` as shown below.



JARs and class folders on the build path:

rhapsody.jar - C:\Rhapsody\8.3.1\Share\JavaAPI

Source attachment: (None)

Javadoc location: (None)

External annotations: (None)

Is not modular - non modifiable

Native library location: C:/Rhapsody/8.3.1/Share/JavaAPI

Access rules: (No restrictions)

Visible only for test sources: No

JRE System Library [JavaSE-1.8]

Figure 8: Project Configuration - Native library location

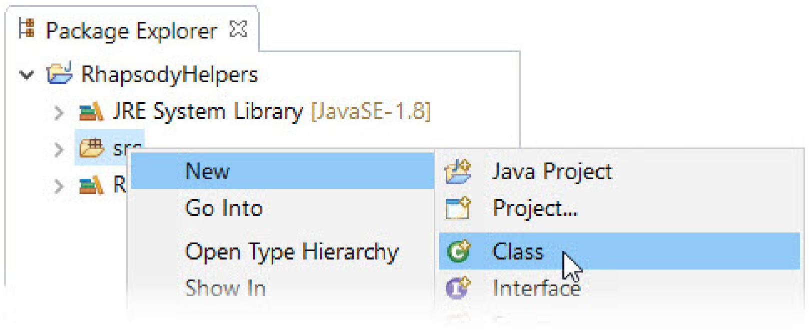
Attention: **Depending on your Rhapsody installation the `/Share` folder might be located in on one of the following locations:

- `C:\Users\<user name>\IBM\Rational\Rhapsody\Share\`
- `C:\<`*`RhapsodyInstallFolder`*`>\Share\`

e) Click **Finish>**

3. Add an initial Java Class:

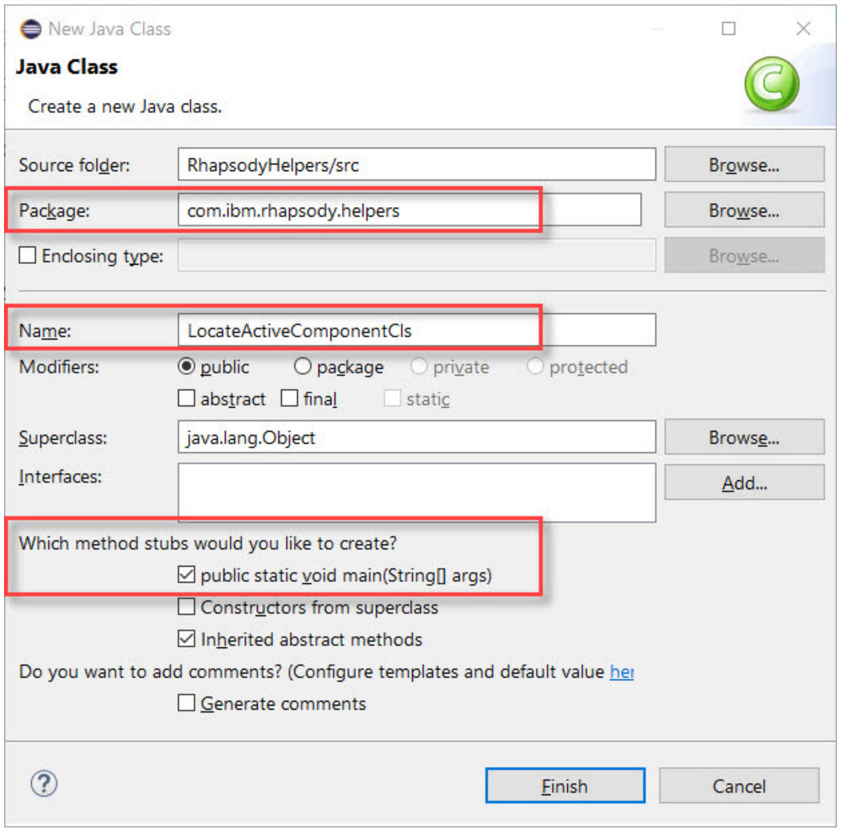
a) Right-click the `src` folder and select **New > Class**



b) Set the **Package** to: `com.ibm.rhapsody.helpers`

c) Set the **Name** to: `LocateActiveComponentCls`

d) Select : **Create public static void main**


New Java Class

Java Class

Create a new Java class.

Source folder: RhapsodyHelpers/src

Package: com.ibm.rhapsody.helpers

☐ Enclosing type:

Name: LocateActiveComponentCls

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value her

☐ Generate comments

Figure 9: Java Class Configuration

e) Click **Finish>**

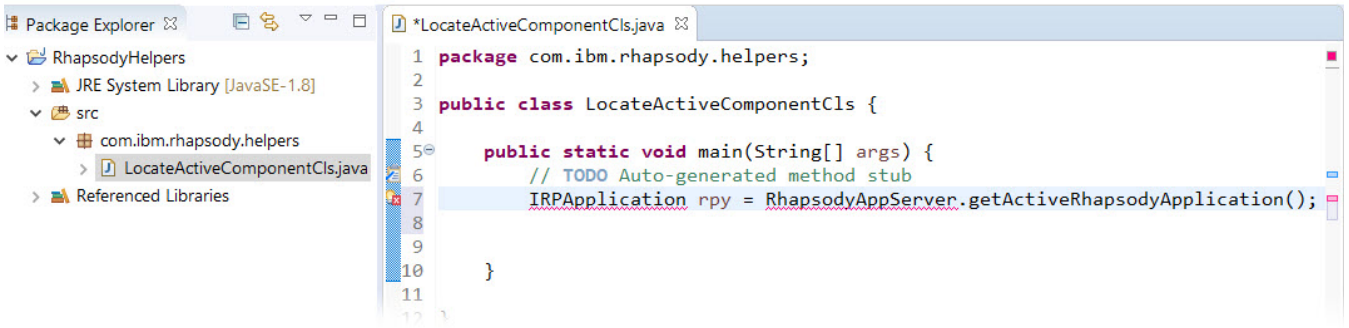
Note: Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc. Please follow this convention during the example. Later you can define your own packages convention to bundle group of related classes/interfaces, etc.

4. Create initial Code:

First we have to connect to the current Rhapsody application. For this we use a function in the Rhapsody library to get the current Rhapsody instance.

a) Add the following code to connect to the current Rhapsody instance

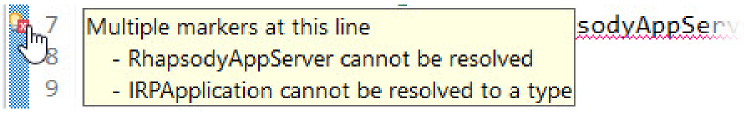
```
IRPApplication rpy = RhapsodyAppServer.getActiveRhapsodyApplication();
```


Package Explorer
RhapsodyHelpers
JRE System Library [JavaSE-1.8]
src
com.ibm.rhapsody.helpers
LocateActiveComponentCls.java
Referenced Libraries
*LocateActiveComponentCls.java

```
package com.ibm.rhapsody.helpers;

public class LocateActiveComponentCls {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        IRPApplication rpy = RhapsodyAppServer.getActiveRhapsodyApplication();


    }
```

Figure 10: Initial Project Code

Note: Eclipse might highlight that there are errors:

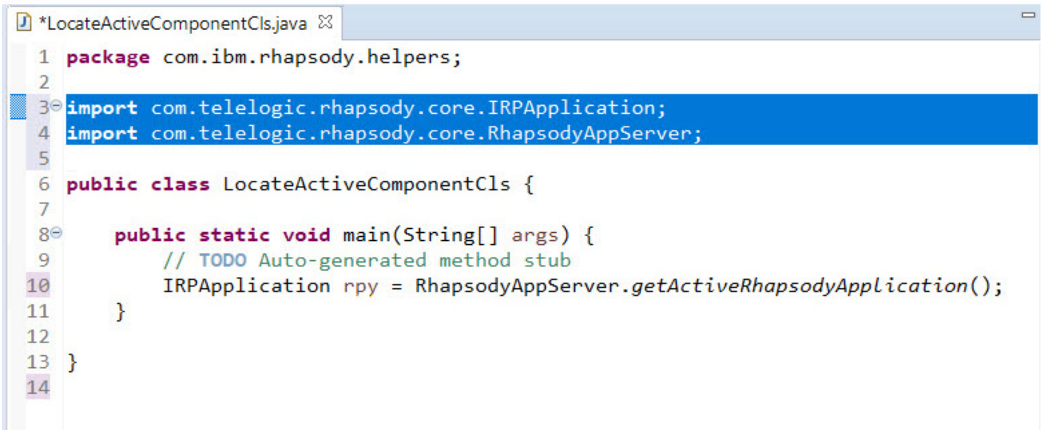


```
Multiple markers at this line
  - RhapsodyAppServer cannot be resolved
  - IRPApplication cannot be resolved to a type
```

Figure 11: Unresolved Types Error

5. Solve unresolved types:

a) Left click on the icon to import `RhapsodyAppServer` in a first step and then `IRPApplication` in a second step.



```
package com.ibm.rhapsody.helpers;

import com.telelogic.rhapsody.core.IRPApplication;
import com.telelogic.rhapsody.core.RhapsodyAppServer;

public class LocateActiveComponentCls {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        IRPApplication rpy = RhapsodyAppServer.getActiveRhapsodyApplication();
    }

}
```

Figure 12: Code after import of unresolved references

6. Calling further Rhapsody API code:

From the connected Rhapsody Application we retrieve its *Active Project*. From this active project we can get its *Active Component* that we're going to locate in the Rhapsody browser

a) Add the following line

```
IRPProject prj = rpy.activeProject();
prj.getActiveComponent().locateInBrowser();
```

b) Left Click on the icon to import unresolved elements again

c) Save the project

Note: This builds the Java Class files automatically:

```
C:\TrainingTools\RhapsodyHelpers\bin\com\ibm\rhapsody\helpers
\LocateActiveComponentCls.class
```

And the folder structure:

- **WorkSpace:** `C:\TrainingTools`
- **Initial Class Folder:** `\RhapsodyHelpers\bin\`
- **Package:**`com\ibm\rhapsody\helpers`

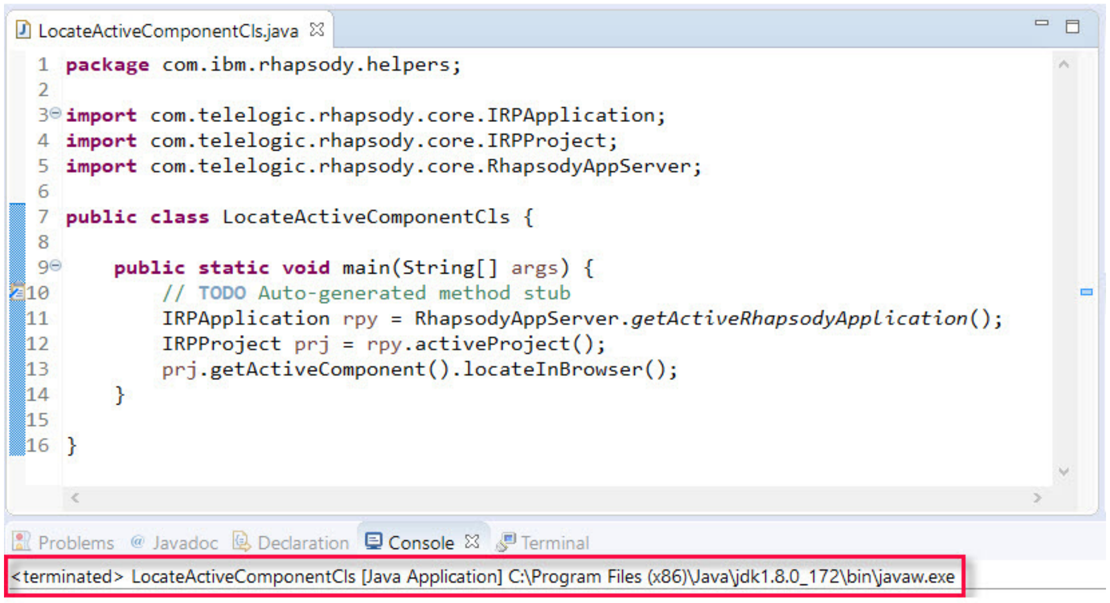
7. Test the Application

Before we integrate and call our application code inside Rhapsody, we will test it in the Eclipse Environment

a) Open Rhapsody and create an initial Project by keeping all settings default

b) In Eclipse press **Run** or **CTRL+ F11**

The code executes and then terminates. The current active Component is selected in Rhapsody



```java
package com.ibm.rhapsody.helpers;

import com.telelogic.rhapsody.core.IRPApplication;
import com.telelogic.rhapsody.core.IRPProject;
import com.telelogic.rhapsody.core.RhapsodyAppServer;

public class LocateActiveComponentCls {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        IRPApplication rpy = RhapsodyAppServer.getActiveRhapsodyApplication();
        IRPProject prj = rpy.activeProject();
        prj.getActiveComponent().locateInBrowser();
    }

}
```

```
<terminated> LocateActiveComponentCls [Java Application] C:\Program Files (x86)\Java\jdk1.8.0_172\bin\javaw.exe
```

Figure 13: Executed and Terminated Eclipse Application

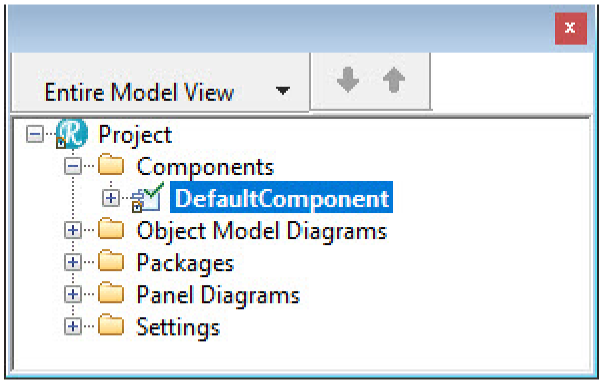


Figure 14: Expanded Rhapsody Browser and its located Active Component

Task 2: Create Rhapsody Helper Menu Entry

To run the Java program from within Rhapsody environment you can extend Rhapsody by adding a helper which will invoke your program. In this Task we will add a new Helper Entry in Rhapsody to launch our application from the Project Menu.

1. Create and add a new Helper menu-entry into the Rhapsody Toolbar:

a) In Rhapsody select **Tools->Customize->Helpers**

b) In the upcoming *Helpers Window* select **New** from the **Menu content** and type `Locate Active Component`

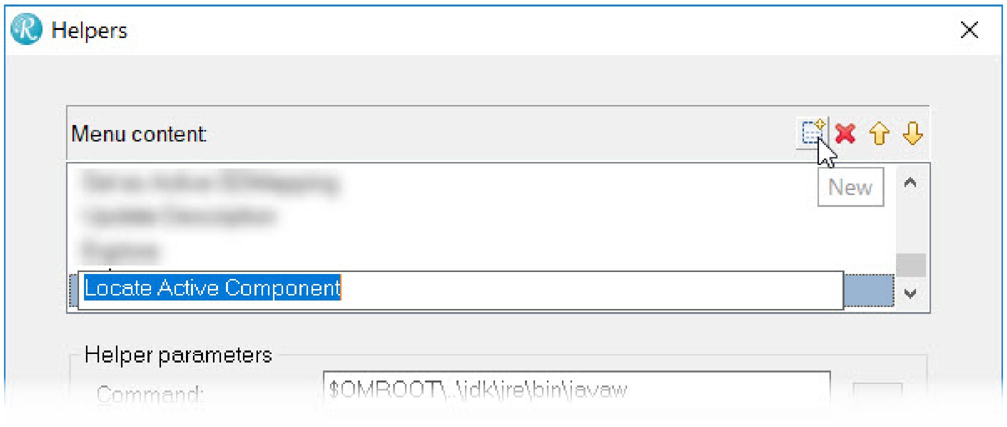


Figure 15: Adding new Menu Entry

2. Adding Helpers Parameter:

a) In the **Command** field add the following entry to run Java.

```
$OMROOT\..\jdk\jre\bin\javaw
```

$OMROOT is a Rhapsody Variablen and points to `RhapsodyInstall/Share` folder.

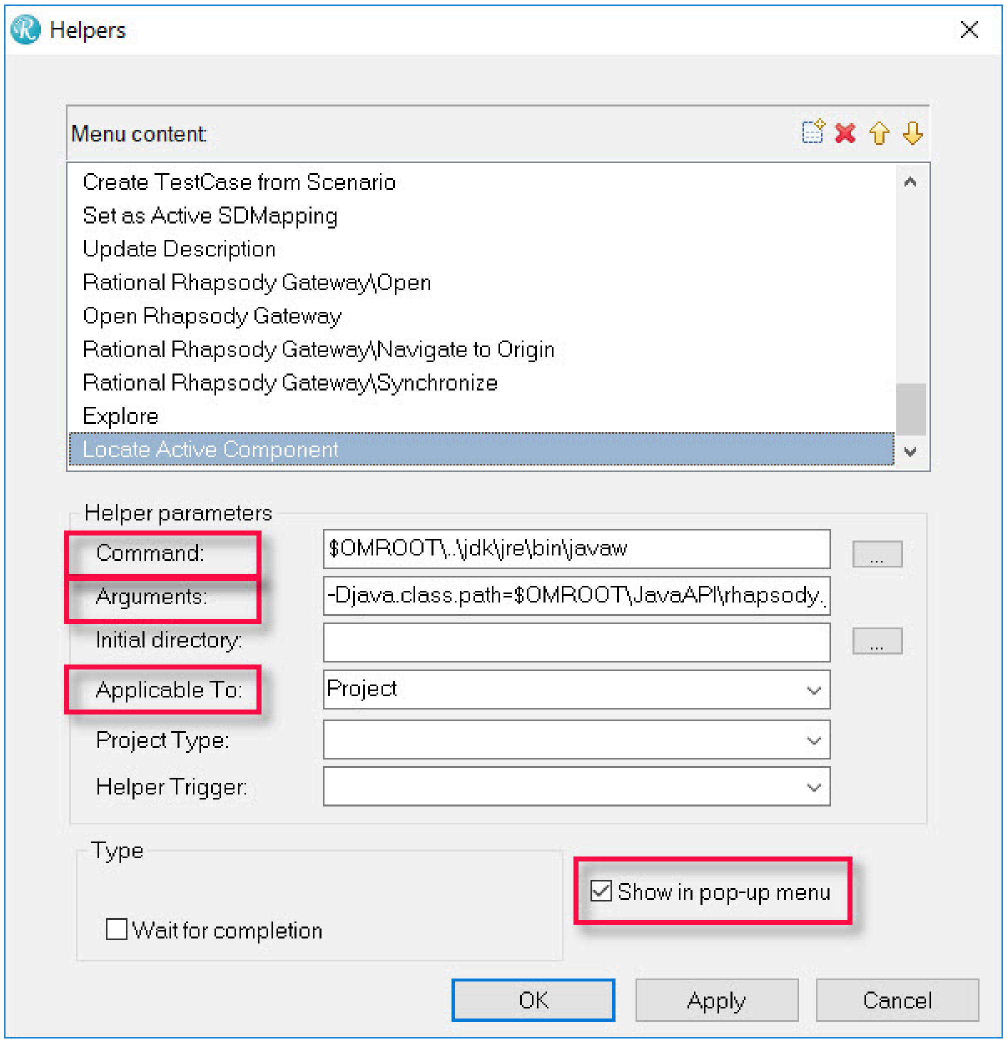
b) In the **Arguments** fild provide the following information:

```
-Djava.class.path=$OMROOT\JavaAPI\rhapsody.jar;C:\Rhapsody-Tools
\LocateActiveComponentPrj\bin
-Djava.library.path=$OMROOT\JavaAPI
 com.ibm.rhapsody.tools.LocateActiveComponentCls
```

CLASSPATH is the path that the Java runtime environment searches for classes and other resoirces (*.jar, *.zip,*.class)

c) In the **Applicable To** select **Project.**

d) Make sure that the **Show in pop-up menue** check-box is selected before you continue.



Helpers

Menu content:

- Create TestCase from Scenario
- Set as Active SDMapping
- Update Description
- Rational Rhapsody Gateway\Open
- Open Rhapsody Gateway
- Rational Rhapsody Gateway\Navigate to Origin
- Rational Rhapsody Gateway\Synchronize
- Explore
- Locate Active Component

Helper parameters

Command:	$OMROOT\..\jdk\jre\bin\javaw
Arguments:	-Djava.class.path=$OMROOT\JavaAPI\rhapsody.
Initial directory:	
Applicable To:	Project
Project Type:	
Helper Trigger:	

Type

☐ Wait for completion

☑ Show in pop-up menu

Figure 16: New Menu Entry - Helpers Paramater

e) Press **OK>** to close the Helpers Window

3. Test the new Helper:

The new helper is available as selection in the project browser.

a) Right click in the project browser and select **Locate Active Component.**

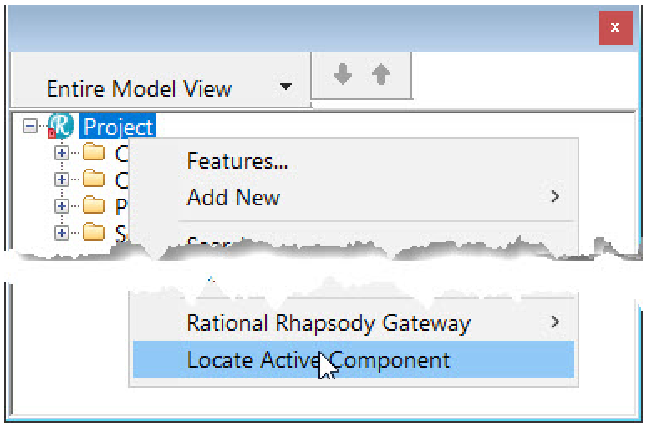


Figure 17: Launch application through context menu

b) Make sure that the current active component is getting selected.

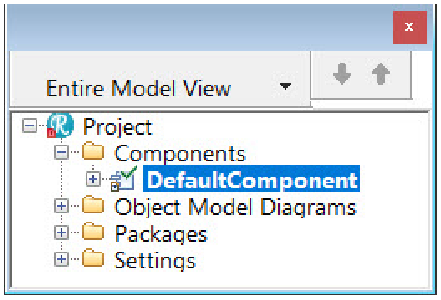


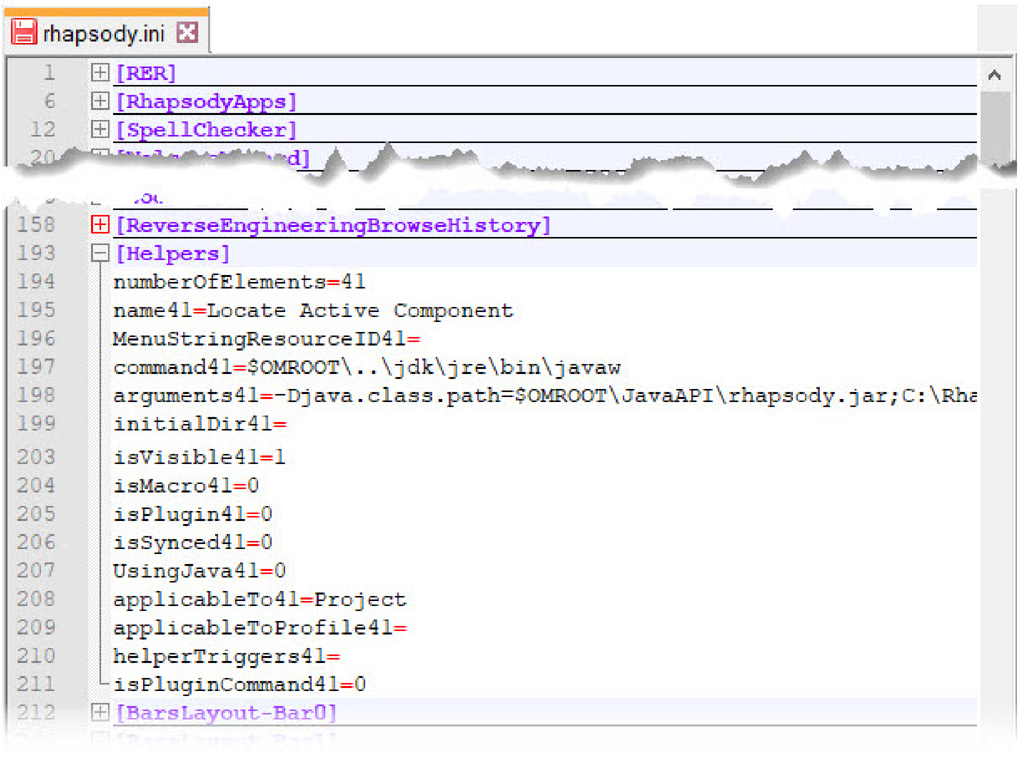
Figure 18: Current Active Component successfully located

4. Observing the `Rhapsody.ini` File:

   The code for your links is added to the `Rhapsody.ini` file.

   a) Open the `Rhapsody.ini` File which is located in the Rhapsody Installation folder.

   b) Search for the entry `Locate Active Component` and observe the text underneath.



```
rhapsody.ini
[RER]
[RhapsodyApps]
[SpellChecker]
[ReverseEngineeringBrowseHistory]
[Helpers]
numberOfElements=41
name41=Locate Active Component
MenuStringResourceID41=
command41=$OMROOT\..\jdk\jre\bin\javaw
arguments41=-Djava.class.path=$OMROOT\JavaAPI\rhapsody.jar;C:\Rha
initialDir41=
isVisible41=1
isMacro41=0
isPlugin41=0
isSynced41=0
UsingJava41=0
applicableTo41=Project
applicableToProfile41=
helperTriggers41=
isPluginCommand41=0
[BarsLayout-Bar0]
```

Figure 19: Extract from rhapsody.ini File- Helpers Entry