Embedded CAL User's Guide

Contributors: Andrew Eisenberg, Greg McClement
Last modified: September 25, 2007

Contents

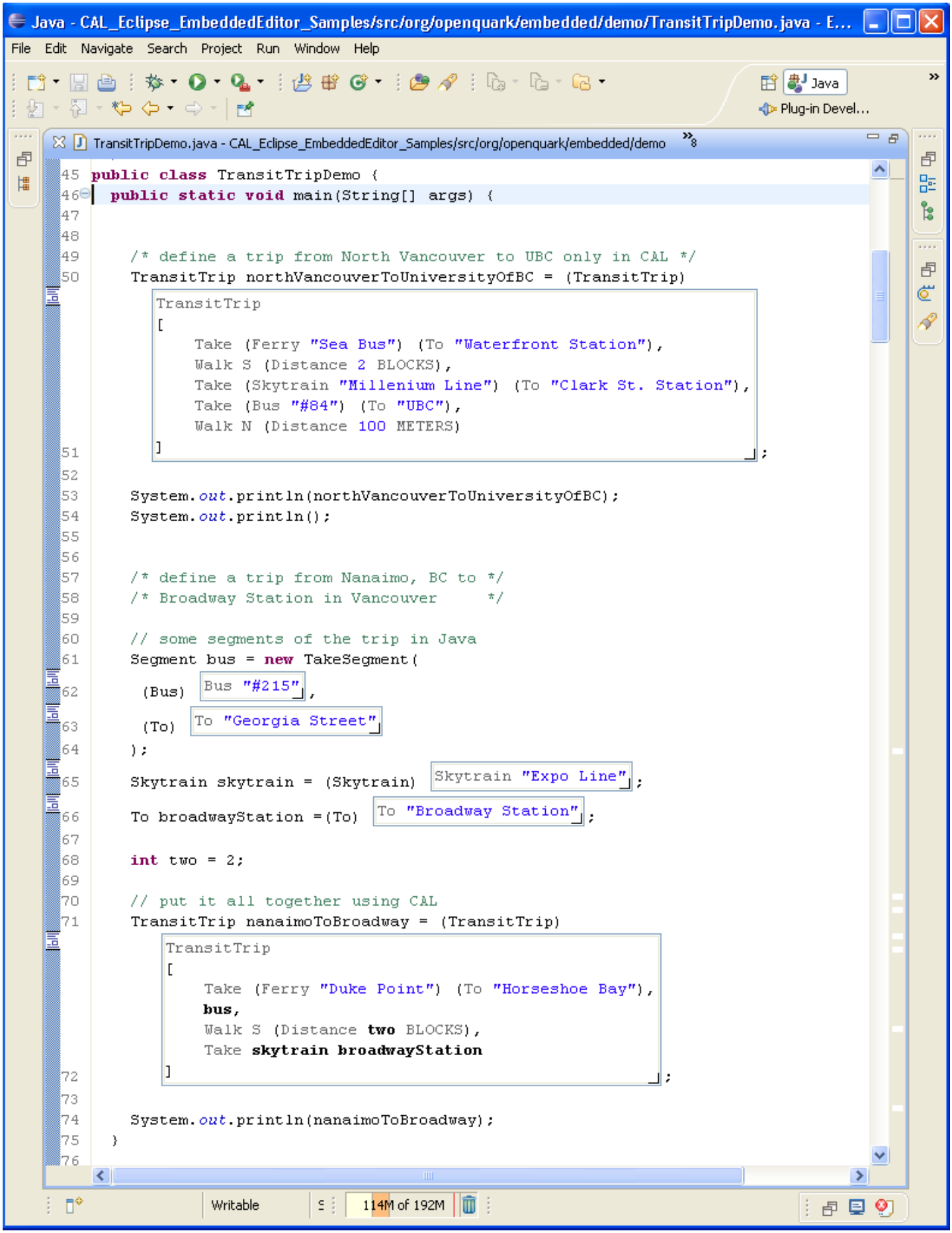
Embedded CAL is an extension to the CAL Eclipse Plug-in that allows programmers to embed CAL snippets into Java classes through a customized editor.

Embedded CAL allows programmers to enter CAL expressions directly into Java code. This means that it is possible to write programs that use both CAL and Java in a single editor. It allows programs that use both CAL and Java to be more concise, easier to read, and to use less glue code when switching between languages. Additionally, through Embedded CAL, it is now possible to pass around lazily evaluated, higher-order functions in Java.

Embedded editors are essentially text widgets contained in a larger, Java editor. These embedded text widgets behave like a standard CAL editor in terms of text highlighting, indentation, text hovers, navigation, etc. The Java editor that contains the CAL editor (the *containing editor*) behaves like a standard Java editor, except for all of the regions covered by embedded editors.

A very first example

I explain Embedded CAL through the following screenshot. Before reading below, please take a few seconds to look at the screenshot. If I have done my job right, the next paragraph is not even necessary. My hope is that what you are seeing is largely intuitive.



If you are familiar with Java, it should be fairly apparent what is going on. Without knowing anything about CAL, it looks like the stuff that is outside the boxes is standard Java and the stuff that is inside the boxes is something special that describes simple trips on public transit (in the Vancouver, BC area). Actually, any CAL expression can be placed inside the boxes, and the CAL expression can contain any Java variables that are in scope. When the program comes across one of these boxes, the expression is evaluated and its value is returned as a Java object. In the first box there is only CAL. But subsequent boxes are smaller and bits of CAL are built up using pieces of Java (and vice versa). As you might expect, outside the boxes, editing feels like a standard Java editor, and inside the boxes editing feels like a standard CAL editor.

Elsewhere, but not shown, I define a simple set of data types in CAL that can be used to describe these trips. I also define input and output policies so that the CAL data types can be converted to Java objects and vice versa. This part is straightforward (but verbose) CAL, which I have not attached because it is not directly relevant to Embedded CAL, but would be happy to share if you are interested.

This first example makes trivial use of CAL itself. I will be adding more examples later on that are more complicated. For example, I will show how you can generate and execute SQL queries in CAL and pass the results back to Java. I will also show how you can use Embedded CAL to pass around anonymous functions in Java.

When stored on disk an embedded editor is a Java method invocation wrapped in special tags. The containing Java editor knows how to parse this and convert it into an embedded editor. If you use a standard text editor to open a Java file that uses embedded editors, you will see the markers `"/* <-- */"` and `"/* --> */"` surrounding all references to embedded editors. The method invocations themselves are calls into the `RunQuark` class (part of the Embedded CAL runtime), which manages all CAL calls from embedded editors.

Hence, Java classes that use embedded editors are simply standard Java classes, and no special compiler is required. Additionally, it is not a requirement to use the Embedded CAL editor to edit these files, it is simply a convenience. It is possible to incrementally introduce Embedded CAL into an existing program.

What is going on here? How does this work? Can I try this out?

The best way to answer the first two questions is to check out the videos (download only, not streaming):

http://resources.businessobjects.com/labs/cal/embedded_cal/Embedded-CAL-Demo1.mov
http://resources.businessobjects.com/labs/cal/embedded_cal/Embedded-CAL-Demo2.mov
http://resources.businessobjects.com/labs/cal/embedded_cal/Embedded-CAL-Demo3.mov
http://resources.businessobjects.com/labs/cal/embedded_cal/Embedded-CAL-Demo4.mov

Or, if you prefer streaming video (over a slower connection), you can use these links instead:
http://homepage.mac.com/luke_e/Andrew/iMovieTheater23.html
http://homepage.mac.com/luke_e/Andrew/iMovieTheater24.html
http://homepage.mac.com/luke_e/Andrew/iMovieTheater25.html
http://homepage.mac.com/luke_e/Andrew/iMovieTheater26.html
Those who just want to cut to the chase may want to skip right to the last one, which offers a fully worked scenario and discusses some more advanced features.

The answer to the last question is a little more complicated. At this point, the plan is to release the feature soon as an optional component of the CAL Eclipse Plug-in. The feature is still experimental and there are still quirks, but it is usable. Installation instructions will be distributed with the release.

Set up & Install

Installation:

1. CAL Eclipse Plug-in (installed as an Eclipse feature). The update site is here:
   http://resources.businessobjects.com/labs/cal/cal_eclipse_update/site.xml
2. Quark Binaries (imported as a project into your Eclipse workspace, see Using CAL with Eclipse:
   http://resources.businessobjects.com/labs/cal/using_cal_with_eclipse.pdf).
   The Quark Binaries project can be downloaded here:
   http://resources.businessobjects.com/labs/cal/open_quark_platform.zip .
   And then imported into the Eclipse workspace by clicking
   File->New->Project->CAL->Quark Binaries Project.
3. Download the Embedded CAL Eclipse Plug-in from the CAL website. The source version is at:
   http://resources.businessobjects.com/labs/cal/embedded_cal/1.7.0/CAL_Eclipse_Embedded_Editor-1.7.0_0-src.zip

   The binary only version is available at:
   http://resources.businessobjects.com/labs/cal/embedded_cal/1.7.0/CAL_Eclipse_Embedded_Editor-1.7.0_0.zip

   You can download either version of the plug-in depending on whether you want to see the source code of the Embedded Editor.
   Unzip the plug-in into the Eclipse plug-in directory so that the org.openquark.cal.eclipse.embedded_1.7.0.0000v20071026 directory is in the Eclipse\plugins directory.

Installing the Embedded CAL Samples Project

This project contains sample code that uses Embedded CAL. You may use it as the starting point for learning about Embedded CAL.

1. Download the Embedded CAL Samples Project from the CAL website http://resources.businessobjects.com/labs/cal/embedded_cal/1.7.0/CAL_Eclipse_Embedded_Editor_Samples-1.7.0_0-src.zip. This is in the form of a zip file.
2. Import the zip file into Eclipse:
   File->Import->Existing Projects Into Workspace->Select Archive File [select the file just downloaded]->Finish.
3. Ensure that the reference to the Quark binaries project is correct. The project assumes that the Quark binaries project is called *Quark*, but if you have a different name, then change it here. Select the project, and then on the menu bar click **Project->Properties->Java Build Path->Projects**. Remove the invalid project and add your own Quark binaries project.
4. Rejoice!

This will import the Embedded CAL Samples project into your workspace. At this point you can try running sample programs that were built using the embedded editor. There are five such programs, Demo1.java, Demo2.java, Demo3.java, Demo4.java and TransitTrip.java .

Creating a project that is aware of Embedded CAL

1. Create a Java Project
2. Add CAL nature
3. Add a reference to the Quark binaries project: Select the project, and then on the menu bar click **Project->Properties->Java Build Path->Projects**.
4. Add the Embedded CAL runtime using the context menu command (In package explorer, right-click on the project, select CAL Tools, and select *Add embedded CAL runtime*

Editing Java files

You will know that Embedded CAL has been properly installed when you open a Demo*.java file from the samples project and you see embedded editors.

If this does not happen, then either Embedded CAL has not been properly installed, or you must explicitly open your file in the Embedded CAL editor. There are 2 ways of doing this:

1. **Window->Preferences->General->Editors->File Associations**.
   Select `*.java`. In the Associated Editors pane, set Embedded CAL as default. This means that all Java files will be open in the CAL Embedded editor by default. (This way is preferred since it will affect all files.)
2. When opening a Java file, right-click and go to **Open With**. Select Embedded CAL Editor.

Embedded Editors

There are two kinds of embedded editors: *Module Embedded Editors* (contains module declaration and import statements and top-level declarations) and *Expression Embedded Editors* (contains a single CAL expression). A class may have no more than one Module Embedded Editor, but any number of Expression Embedded Editors.

Both kinds of embedded editors can be placed anywhere a Java expression may go. The return value of a Module Embedded Editor can be ignored (it is always null). The return value of Expression Embedded Editors is the value of the expression.

To add an embedded editor to a class, you can use the context menu. Two menu items are provided: *Insert Embedded Module* and *Insert Embedded Expression*. Or, you can add an embedded editor by typing the characters `"/* <-- */ /* --> */"` and then saving the file.

Features

The following features are available in both kinds of embedded editors

Auto-indent

The cursor will indent properly (according to the CAL preferences) in either kind of embedded editor after a carriage return or tab is pressed.

Syntax highlighting

Syntax highlighting in an embedded editor uses standard CAL highlighting with the following changes.

Java variables used in an expression editor are in bold.

Names imported from outside modules will be italicized blue.

Code Completion

Simple code completion exists. It captures all top-level declarations, all items that are explicitly imported, and all java variables that are currently in scope (for expression

editor only). Note that this code completion is not context sensitive as you would find in a standard CAL editor (more on this later).

Hover

Hovering will show the CALDoc (or the source code if Shift is pressed). Identifiers defined locally in an expression editor will have no hover. Hovers on identifiers defined as a top-level declaration in the associated Module editor cannot be made "sticky" (ie- F2 will not work).

Eager parsing

Syntax errors in an embedded editor will turn the border color red. When the error is fixed, the color will return to blue (for expression editors) or green (for module editors).

[ADE- In the future, I would like to see more than just syntax checking. We could also do type checking and name resolution fairly easily. I would also like to see the error itself highlighted, not just the entire box (making it easier to spot where the error is).]

Navigation

By pressing F3 or using the context menu in an embedded editor, it is possible to navigate to the declaration of an identifier. Locally defined identifiers are ignored.

Gutter annotations

Embedded editor annotations appear in both the left and the right side of the editor. Hovering over an annotation will show the first 50 characters of the embedded editor as a preview. Clicking on an annotation on the right side will reveal and give focus to that embedded editor.

What modules are in my CAL workspace? or How can I control which modules I have access to?

(See the document Java Meets Quark http://resources.businessobjects.com/labs/cal/java_meets_quark.pdf (pages 5--6) for a full discussion of workspaces and the `BasicCALServices` Java class.)

By default, embedded editors use the `cal.samples.cws` workspace. This includes most non-test modules in the Quark Binaries project.

If you would like to use your own modules, then you must create your own workspace file. The easiest way to do this is to start with an existing workspace file and adapt it for your own purposes. You can include any extra module that you require.

Once this is done, you must add the following code to your program and it must run before any other Embedded CAL code is executed:

```
BasicCALServices services =
BasicCALServices.make(<workspace_name>);
RunQuark.init(services);
```

Expression Editor

Expression editors contain a single CAL expression. The program treats the editor as a function. Every Java variable used in the editor is treated as an argument. Input policies are inferred based on the runtime type and output policies can be explicitly specified.

Java variable capture

Any java variable that is in scope can be used in the expression editor. You may type it in.

Input policy

Input policies are inferred based on runtime type. Primitive types and Strings use the associated input policy. Arrays or lists of a primitive type or String use the appropriate input policy for a list.

If the precise input policy cannot be inferred, then the DEFAULT_INPUT_POLICY is used.

[ADE- I would like to be able to specify a particular input policy. Also, lists of lists are not handled.]

Error handling

By default, all errors are caught by the Embedded CAL runtime library. However, this can be overridden.

- Right-click on embedded editor
- Select "Show expanded view"
- Check the "Throws Exception" box
- Now, the editor's exception is not caught by the library, but must be explicitly handled by the caller.

At this point, you may uncheck *Show expanded view* for a more concise view of the editor.

Output policy

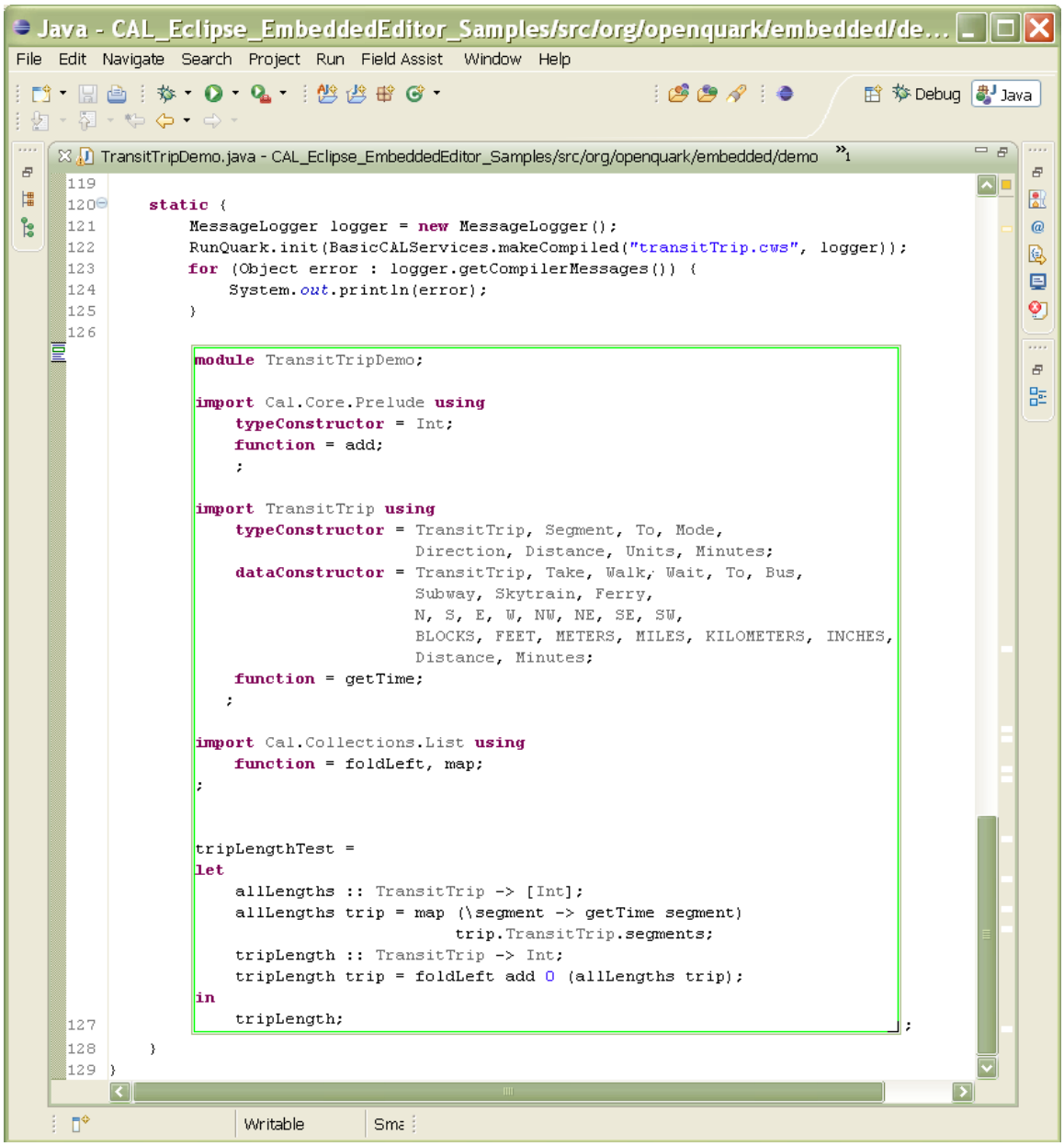
By default, a `DEFAULT_OUTPUT_POLICY` is used, but this can be overriden.

- Right-click on embedded editor
- Select *Show expanded view*
- The drop-down menu controls the output policy. Select `DEFAULT_OUTPUT_POLICY, CALVALUE_OUTPUT_POLICY`, or you may type in any expression that has a type of `OutputPolicy`. Leaving the drop-down blank, or selecting `<<infer output policy>>` will use the `DEFAULT_OUTPUT_POLICY`.

Module Editor

All expression editors run in the context of a module. If no module editor is present, then expression editors run in a default module that imports `Cal.Core.Prelude` and nothing else. If you require using more than the `Prelude` module, then you must use a module editor.

The following screen shot shows a module editor. They are distinguished from expression editors by being highlighted in green.



Each Java compilation unit may have no more than 1 module editor. A warning will appear if you try to enter a second one.

The module editor should go in a location that is executed only once and executed before any expression editors are. In the above example, therefore, the module editor is placed in a static initializer. Also, a module editor evaluates to a Java expression and so, it is necessary to put a semi-colon after the editor.

The code above the module editor specifies a custom workspace file. You can use this code to specify a workspace other than the default:


```
MessageLogger logger = new MessageLogger();
RunQuark.init(
     BasicCALServices.makeCompiled(
          "transitTrip.cws", logger));
for (Object error : logger.getCompilerMessages()) {
     System.out.println(error);
}
```

Oddities and Errors

Here are a list of known bugs and oddities. Please be patient as embedded CAL is still in its beginning stages.

- When entering or exiting editors by clicking with the mouse, the screen sometimes scrolls to a new location. This seems to happen more often with embedded editors that take the entire vertical part of the screen.
- The Java text behind the embedded editor is not formatted into nice Java, but rather it is simply stored on one line. The reason for this is that it is difficult to hide text that spans multiple lines due to a lack of support for this in the SWT `StyledText` widget.
- If you do ever want to view Java code with embedded CAL in a standard text editor, you may format it any way you like. When this code is again displayed in an editor that supports embedding, it will automatically be reformatted so all hidden code is on one line.
- There are several ways to delete an embedded editor:

    - Right-click on editor and select "Delete Editor"
    - Select a region of Java code that includes an editor and Cut it (ctrl-X in windows). (The editor can later be pasted (ctrl-V in windows) elsewhere.)
    - Delete line (ctrl-D in windows)

    The following methods will *not* work:
    - Using the backspace or delete key

- Sometimes, when the caret is placed in the Java world just on the edge of an embedded editor, typing will place characters *inside* of the embedded editor (whereas the correct behavior is that the characters should appear *outside* of it). If this happens, just be aware of it and try typing one character away from the edge of the embedded editor.
- Keyboard combinations for copy and cut do not work inside of an embedded editor. You must use the menu for this. However, keyboard command for paste works fine.
- Undo does not remove the embedded editor after an insert, but instead it removes some characters.
- Content assist is not context aware. It merely looks at the prefix and matches to all possible completions. It does not filter completions with respect to context.

- When pressing ctrl-home to go to the top of a file, the current embedded editor stays on the screen.

If you have any problems or issues with Embedded CAL, or you find any errors in this document, there is a Google Group on the CAL Language and Quark Platform. This is the primary place for discussions, announcements, and questions about CAL and Quark. In addition, please feel free to send comments or questions to the primary author, Andrew Eisenberg (andrew@eisenberg.as).

