

Using CAL with Eclipse

**Rick Cameron
Andrew Eisenberg
Edward Lam
Greg McClement**

Using CAL with Eclipse

by Rick Cameron, Andrew Eisenberg, Edward Lam, and Greg McClement

Last modified: August 24, 2007

Copyright (c) 2007 BUSINESS OBJECTS SOFTWARE LIMITED
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Business Objects nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Overview

This document explains how to use CAL with Eclipse. It consists of four parts:

Installing the CAL Eclipse Plug-in

Describes how to install the plug-in, either from the update web site or from the archived .zip file

Setting up Eclipse for CAL

Explains how to add the Quark Platform into your Eclipse workspace

Tutorial: Hello World

A brief introduction to using CAL in Eclipse

Using the CAL Eclipse Plug-in

Information on the features of the plug-in

We highly recommend that you install the CAL Eclipse Plug-in. It provides many useful features when editing CAL source code.

System requirements

- Eclipse 3.2 or greater.
- Java 5 or greater.

Quark has been tested by us mainly on Windows XP, Linux and Mac OS X, but as a pure Java application may work on other platforms supporting Java 5 or greater.

Call for feedback

Any feedback you might have to help improve this document is very welcome. Please send any comments, suggestions, or questions to the CAL Language Discussion forum on Google Groups http://groups.google.com/group/cal_language.

Installing the CAL Eclipse Plug-in

You can install the CAL Eclipse Plug-in from the update web site, or from the .zip file included in the Open Quark distribution.

1. From the CAL Eclipse Plug-in update site

- Select from the Eclipse menu: **Help** → **Software Updates** → **Find and Install...**
- Select the "Search for new features to install" radio button. Click "Next".
- Click the "New Remote Site..." button
 - Name: Enter a name for the site. e.g. "CAL Eclipse Plug-in update site"
 - URL: `http://resources.businessobjects.com/labs/cal/cal_eclipse_update/site.xml`
- Click "Finish"
- Select the feature to install. Click "Next".
- Read the license agreement. If you accept, select the radio button and click "Next".
- Click "Finish"

2. From the archived update site

- Download cal_eclipse_update.zip to your local machine.

NB: The .zip file contains a text file with release notes. We advise you to read this file before proceeding, in case the setup procedure has changed.
- Select from the Eclipse menu: **Help** → **Software Updates** → **Find and Install...**
- Select the "Search for new features to install" radio button. Click "Next".
- Click the "New Archived Site..." button
 - Select the site .zip file
- Enter a name for the site. E.g. "CAL Eclipse Plug-in archived update site". Click "OK".
- Click "Finish"
- Select the feature to install. Click "Next".
- Read the license agreement. If you accept, select the radio button and click "Next".
- Click "Finish"

3. Source Code for the CAL Eclipse Plug-in

The Open Quark distribution includes a .zip file containing the source code for the CAL Eclipse Plug-in. You can import these projects into an Eclipse workspace by unzipping the file to a directory and directing Eclipse to import all projects found in the directory.

Warning

Importing the source code is not an alternative to installing the plug-in. You would only import the source code if you intend to study or modify the plug-in. As with all Eclipse plug-in development, changes to the source code are tested by launching a second copy of Eclipse that will load the modified plug-in.

Setting up Eclipse for CAL

In order to use CAL in your Java projects, you must include the Quark Platform in your Eclipse workspace. You can choose between adding the platform in binary form or in source form.

The binary version consists of several .jar files that contain the compiled Java runtime for CAL, together with the CAL modules that make up the core and standard libraries of the platform. It can be included into Eclipse as a single project and provides everything you need to run Quark tools like ICE and GemCutter and to create CAL modules in your own projects.

The source version consists of the same set of components as the binary version, except that everything is provided in source form, including the Java runtime and samples. When imported into Eclipse, it creates several projects. You can use it, as above, to run tools and to create your own CAL modules; but you can also trace into the source of the runtime and experiment with modifications to it.

Most programmers will choose to set up the binary version, as this is simpler and avoids the need to compile the source of the runtime.

NB: Once you have installed the CAL Eclipse Plug-in, you must add the CAL nature to any project that contains CAL modules to have those modules recognized by the plug-in. You can do this by using the CAL Tools submenu of the context menu for the project in the Package Explorer. If you use the New CAL Module wizard to make CAL Modules then the nature will be added automatically.

After the setup instructions you will find a short tutorial on various ways to run a CAL function from within Eclipse.

1. Adding the Quark Platform Binaries to Your Workspace

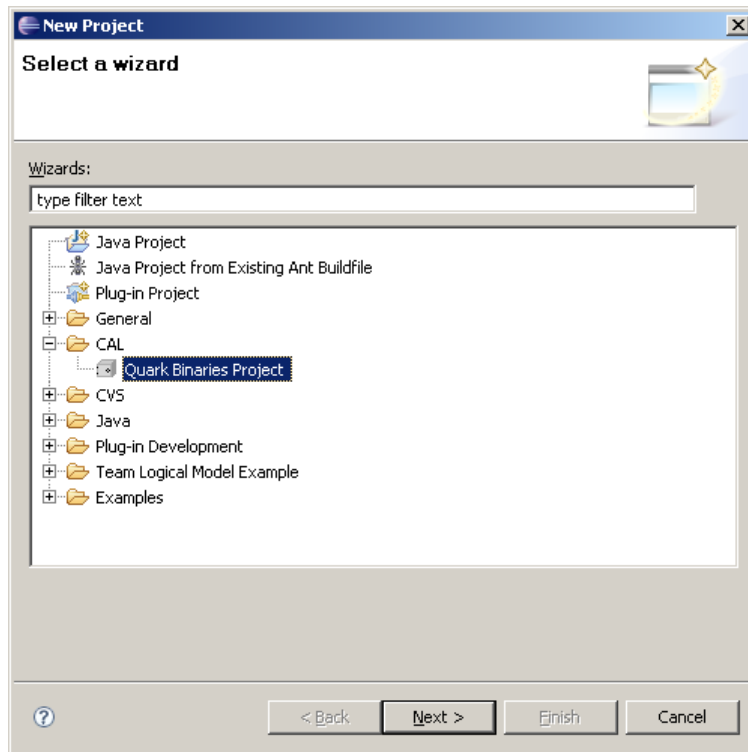
To create a Java project with the binary form of the Quark Platform:

- Unzip the binary Quark Platform distribution from open_quark_platform.zip

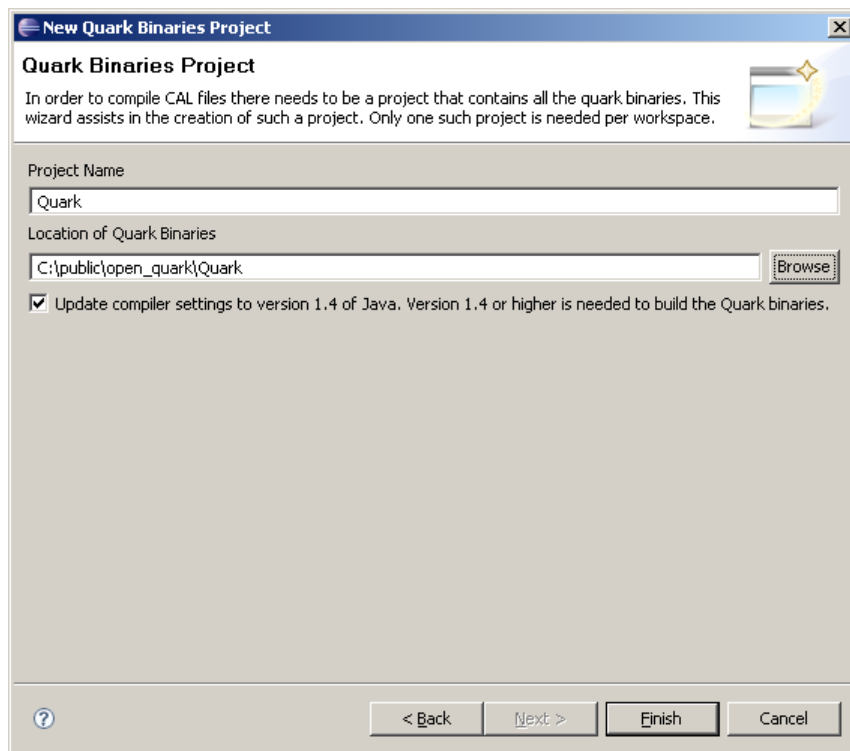
Next there are two options for making the Quark binaries project. This can be done using the CAL Eclipse wizard as described in the next section or using the Java project wizard as described in the following section.

Create a Quark Binaries Project using the CAL Eclipse Plug-in

- In the Eclipse IDE select **File** → **New** → **Project...** and under the CAL category select the "Quark Binaries Project" wizard.



- In the new Quark binaries project wizard, enter a name for the Quark binaries project and the location where you unzipped the downloaded Quark binaries. If the Java compiler compliance level in Eclipse is less than 5.0, then there will be a check box for setting the compiler to a higher version. This will allow the wizard to set the compiler to a version that Quark binaries need.



Create a Quark Binaries Project using the Eclipse Java Project Wizard

- Copy the two root project files (.classpath and .project) from the "eclipse-support" folder to its parent ("Quark") folder of the unzipped Quark Platform distribution.
- If the Java compiler compliance level in Eclipse is less than 5.0, change it to 5.0 (or higher)
 - From the Eclipse menu, select: **Window** → **Preferences...**
 - Preferences dialog:
 - Select (in the preferences tree): **Java** → **Compiler**.
 - Uncheck "Use default compliance settings".
 - Change the following to "5.0": "Compiler compliance level", "Generated .class files compatibility", "Source compatibility"
- From the Eclipse menu, select: **File** → **New** → **Project...**
- Wizard (Select a wizard):
 - Select **General** → **Project**
 - Click "Next".
- Wizard (New Project):
 - Enter a project name (e.g. "Quark").
 - Uncheck "Use default location"
 - Under "Location:", select the "Quark" folder from the unzipped Quark Platform distribution.
- Click "Finish".

Running ICE or GemCutter (using Quark Platform binaries)

- From the Eclipse menu, select: **Run** → **Run...**
- Create a new Java Application configuration
 - Under configurations (on the left pane), right-click "Java Application", select "New"
- Enter a launch configuration name
 - Enter a name in the "Name:" field. e.g. "ICE".
- Configure the Main tab:
 - Set the Project.
 - This should be the project created above.
 - Set the Main class.
 - For ICE: set this to "org.openquark.cal.ICE".
 - For GemCutter: set this to "org.openquark.gems.client.GemCutter".
- Configure the Arguments tab:
 - Specify the heap memory for the JVM:
 - Under VM arguments, add: **-Xmx256m**
- Click "Run".

2. Adding the Quark Platform to Your Workspace as Source Code

To import the Quark Platform into Eclipse as source-code projects:

- Import the projects:
 - Unzip the source Quark Platform distribution from open_quark_platform_src.zip. This places all files under a directory called "Quark".
 - From the Eclipse menu, select: **File** → **Import...**
 - Wizard (Import)
 - Select **General** → **Existing Projects into Workspace**
 - Click "Next"
 - Wizard (Import Projects)
 - Click "Browse..." next to "Select root directory:"
 - Select the Quark directory created above
 - Click OK
 - You will see several projects in the "Projects:" list. All except the "BAM_Sample", "CAL_Samples" and "CAL_Benchmarks" projects are essential parts of the Quark Platform
 - Click "Finish"
- Disable unresolved PDE dependency compilation errors for the import project:
 - Open the Package Explorer view:
 - From the Eclipse menu, select: **Window** → **Show View** → **Package Explorer**.
 - Right-click the "import" project, select "Properties".
 - Properties for import dialog:
 - Select (in the preferences tree): **Plug-in Development** → **Plug-in Manifest Compiler**.
 - Select "Use project settings".
 - On the "Plugins" tab: change "Unresolved dependencies" from "Error" to "Ignore".
- If the Java compiler compliance level is less than 5.0, change it to 5.0 (or higher)
 - From the Eclipse menu, select: **Window** → **Preferences...**
 - Preferences dialog:
 - Select (in the preferences tree): **Java** → **Compiler**.
 - Uncheck "Use default compliance settings".
 - Change the following to "5.0": "Compiler compliance level", "Generated .class files compatibility", "Source compatibility".

Running ICE or GemCutter (using Quark Platform source code)

- From the Eclipse menu, select: **Run** → **Run...**
- Create a new Java Application configuration
 - Under configurations (on the left pane), right-click "Java Application", select "New"
- Enter a launch configuration name
 - Enter a name in the "Name:" field. e.g. "ICE".
- Configure the Main tab:

- Set the Project and Main class.
 - For ICE: use "CAL_Platform" and "org.openquark.cal.ICE".
 - For GemCutter: use "Quark_Gems" and "org.openquark.gems.client.GemCutter".
- Configure the Arguments tab:
 - Specify the heap memory for the JVM:
 - Under VM arguments, add: `-Xmx256m`
- Click "Run".

3. Tips and Tricks

- To find .cal files and .cws files in Eclipse:
 - From the Eclipse menu, select: **Navigate** → **Open Resource...**
 - Open Resource dialog:
 - Type "*.cal" or "*.cws".
- To use a CAL workspace other than the default when launching a project:
 - Open the appropriate launch configuration
 - Configure the Arguments Tab:
 - Under VM arguments, add:


```
"-Dorg.openquark.cal.workspace.spec="StandardVault simple workspace name"
```

 e.g. `-Dorg.openquark.cal.workspace.spec="StandardVault cal.samples.cws".`
 - Configure the Classpath Tab, if the workspace is not in the CAL_Platform project:
 - In the "Classpath:" tree, select "User Entries". Click "Add Projects...".
 - Check the project containing the desired workspace. Click "OK".

Tutorial: Hello World

The following are examples of how to write and call a function in CAL to output "Hello World". The tutorial assumes that you have installed the CAL Eclipse Plug-in.

1. From ICE command line

- Start ICE.
- Enter the code

```
=> "Hello World" (including quotes)
```

2. In an existing module

- Open the file "String_Tests.cal" in Eclipse.
 - From the Eclipse menu: **Navigate** → **Open Resource...**
 - In the dialog box, select "String_Tests.cal".
- Add the function.
 - Navigate to the end of the file. Type:

```
myFunction = "Hello World";
```

- If ICE is running, type ":rc" to recompile modified modules. Otherwise, start ICE. This will automatically recompile modified modules.
- Change to the String_Tests module.

```
=> :sm String_Tests
```

- Run the function.

```
=> myFunction
```

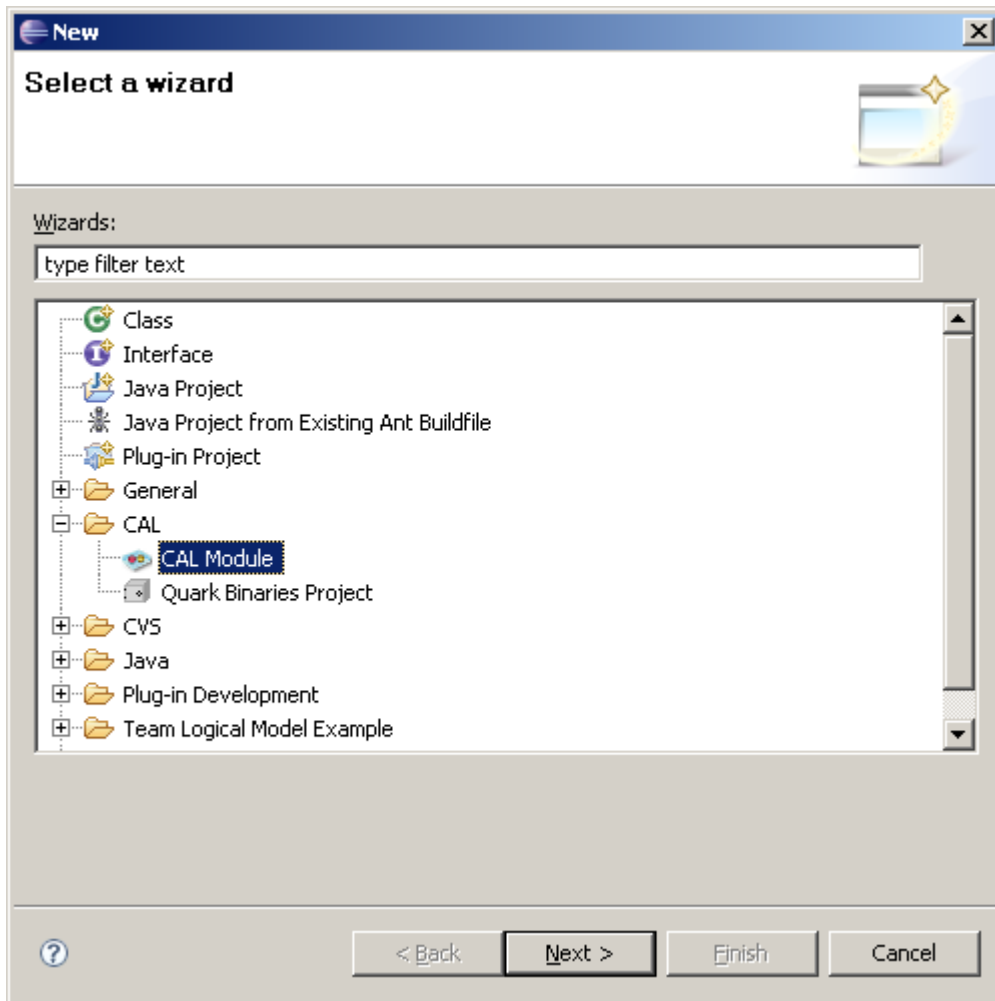
3. In a new module

In this step, you will create a new CAL module.

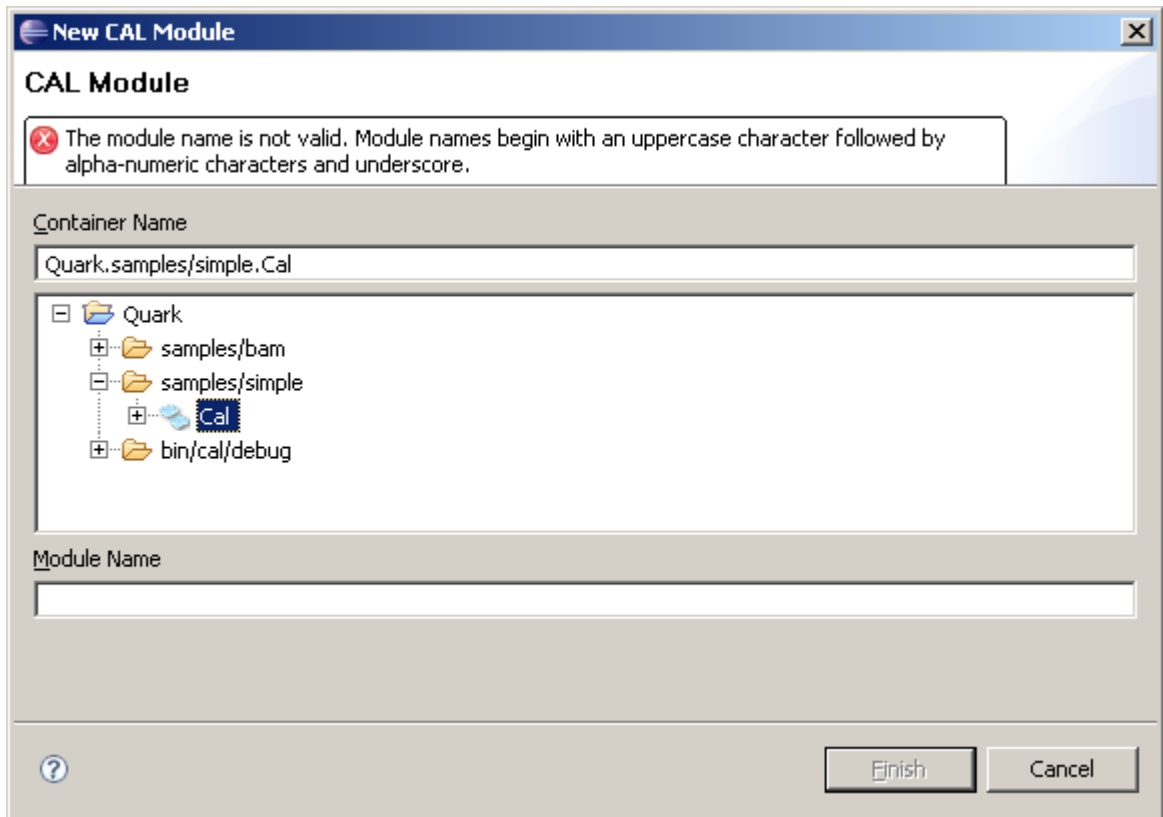
If you installed the Quark Platform binaries, you can create the module in the Quark binaries project.

If you installed the Quark Platform as source projects, you can create the module in the CAL_Samples project. You will also have to add the CAL_Samples project to the classpath of the launch configuration for ICE.

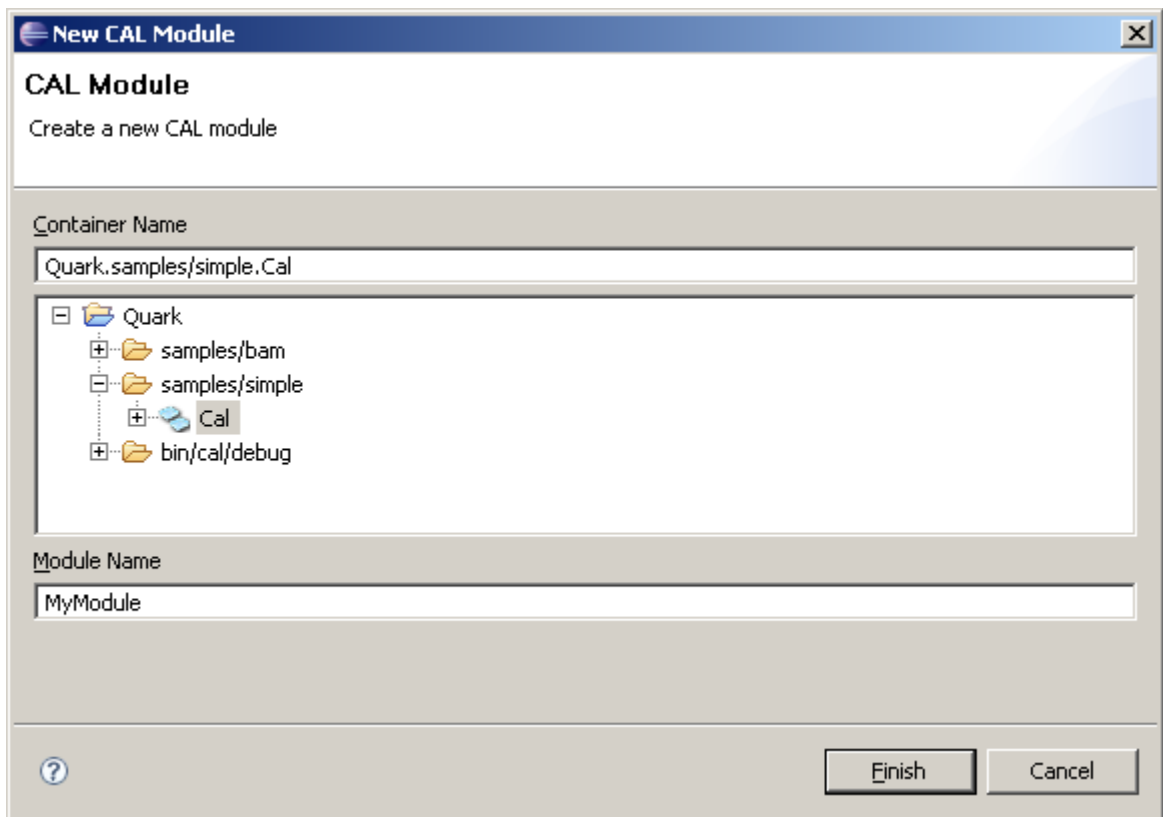
- Create a new module "MyModule" in the appropriate project.
 - From the Eclipse menu: **File** → **New** → **Other...**. In the wizard dialog, select: **CAL** → **CAL Module**



- Select the appropriate CAL project.



- For the file name, enter "MyModule" (case-sensitive).



- Click "Finish"
- In the editor, enter the following text after the import declaration.

```
myFunction = "Hello World";
```

- Save the file
- Run ICE (if not already running).
- Add the module to the current environment.

```
=> :adm MyModule
```

- Change to the new module.

```
=> :sm MyModule
```

- Run the function.

```
=> myFunction
```

4. In a new module in a second project, loaded via a CAL workspace file

- Create a new Java project.
 - From the Eclipse menu, select: **File** → **New** → **Project...**
 - Wizard (Select a wizard):
 - Select "Java Project".
 - Click "Next".
 - Wizard (Create a Java project):
 - Enter a project name
 - Click "Finish".
- Add a Java source folder
 - In the Package Explorer view, right-click on the new project. Select **New** → **Source Folder**.
 - Enter "MySource" as the folder name.
- Create a new module "MyModule2" in the new source folder.
 - In the Package Explorer view, right-click on the source folder. Select **New** → **Other...**. In the wizard dialog, select: **CAL** → **CAL Module**.
 - For the file name, enter "**MyModule2**" (case-sensitive).
- In the editor, enter the following text after the import declaration.

```
myFunction = "Hello World";
```

- Add a folder for workspace declarations
 - In the Package Explorer view, right-click on the new MySource folder. Select **New** → **Folder**.

- For the folder name, enter "Workspace Declarations" (case-sensitive).
- Create a new workspace declaration "myworkspace.cws" in the Workspace Declarations folder
 - In the Package Explorer view, right-click on the Workspace Declarations folder. Select **New** → **File...**
 - For the file name, enter "myworkspace.cws" (case-sensitive).
- Open the new file in an editor
 - Expand the Workspace Declarations folder.
 - Double-click myworkspace.cws.
- In the editor, enter the following text:

```
StandardVault MyModule2
import StandardVault cal.platform.cws
```

Option 1: load the workspace on startup

- Create a new run target for ICE, which includes the new project on the classpath.
 - Follow the instructions above to create a new run target for ICE, with the following modifications:
 - On the Classpath tab, add the new project to the classpath.
 - Select "User Entries" (under the "Classpath:" tree).
 - Click "Add Projects..."
 - Select the new project.
 - On the Arguments tab, specify the startup CAL workspace file.
 - Under VM arguments, add:


```
-Dorg.openquark.cal.workspace.spec="StandardVault myworkspace.cws".
```
- Run the new ICE run target.
- Change to the new module.

```
=> :sm MyModule2
```

- Run the function.

```
=> myFunction
```

Option 2: load the workspace from the ICE command prompt

- Create a new run target for ICE, which includes the new project on the classpath.
 - Follow the instructions above to create a new run target for ICE, with the following modifications:
 - On the Classpath tab, add the new project to the classpath.
 - Select "User Entries" (under the "Classpath:" tree).
 - Click "Add Projects..."
 - Select the new project.
 - Run the new ICE run target.
 - Load the new workspace.

```
=> :ldw myworkspace.cws
```

- Change to the new module.

```
=> :sm MyModule2
```

- Run the function.

```
=> myFunction
```


Using the CAL Eclipse Plug-in

The CAL Eclipse Plug-in makes editing CAL source code easier by offering the following components:

The CAL Editor

An enhanced text editor that provides syntax coloring, source modification commands and refactorings.

The CAL Builder

A CAL language analyzer that runs in the background and enables smart processing of CAL source code.

The CAL Workspace View

An Eclipse view that displays all the CAL modules in the workspace as well as the CAL types and functions defined in the modules.

The Outline View

When the CAL editor is active, the standard Outline view shows an outline of the CAL types and functions in the current CAL module.

The Quick Outline View

The Quick Outline view shows a tree of the types, constructors, type classes and functions in the CAL module being edited.

The CAL Console

This console allows CAL expressions to be entered and evaluated with respect to the CAL modules in the Eclipse workspace.

The CAL Metadata Editor

This editor allows you to view and modify the CAL metadata associated with a CAL entity.

CAL Search

A tab in the Search dialog that allows you to search all CAL modules for definitions or references to an identifier

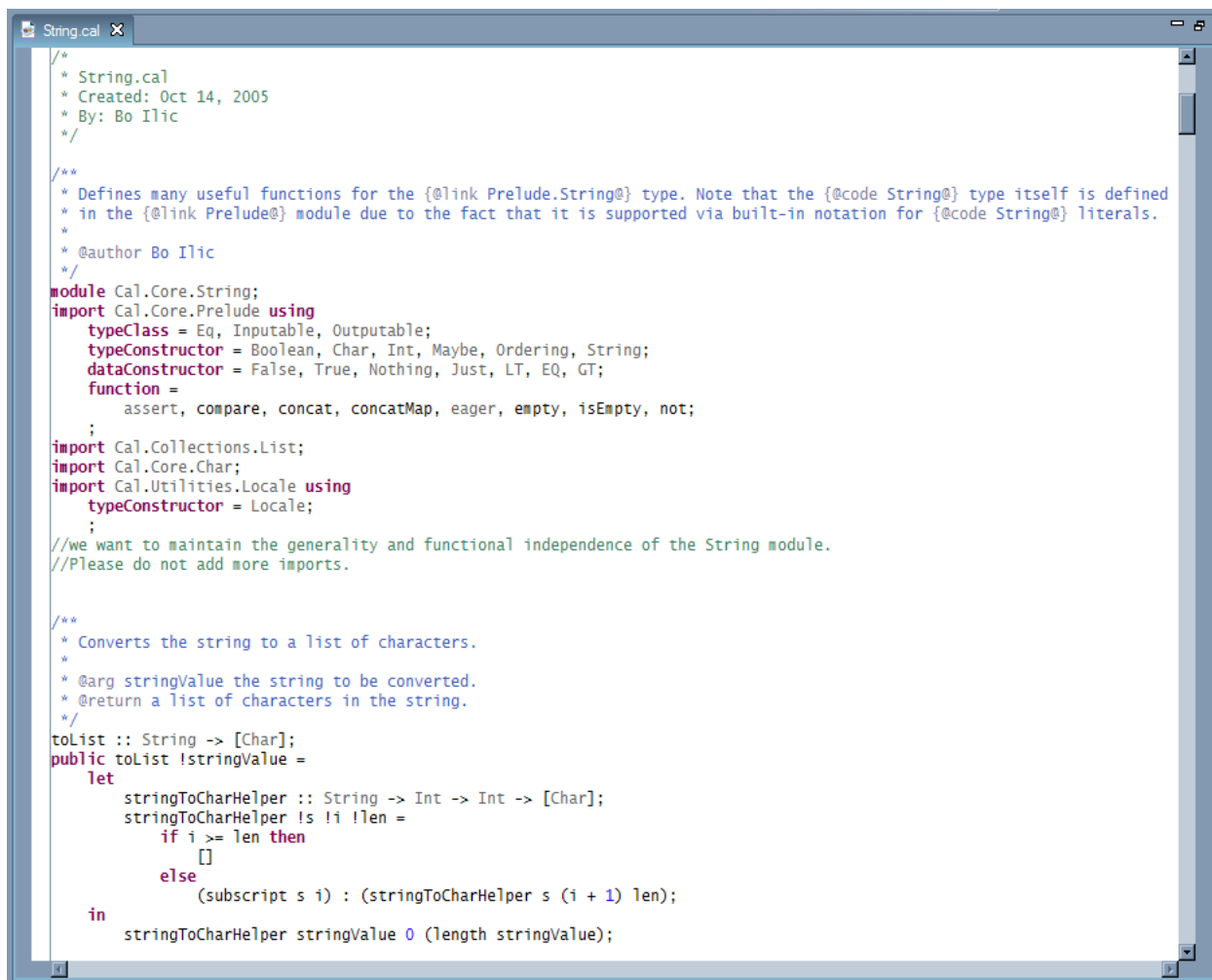
Preferences Pages

There are several pages in the Preferences dialog related to the CAL Editor and Builder.

The CAL Perspective

There is a perspective that is tailored for editing CAL modules

1. The CAL Editor



```
String.cal
/*
 * String.cal
 * Created: Oct 14, 2005
 * By: Bo Ilic
 */

/**
 * Defines many useful functions for the {@link Prelude.String@} type. Note that the {@code String@} type itself is defined
 * in the {@link Prelude@} module due to the fact that it is supported via built-in notation for {@code String@} literals.
 */
@author Bo Ilic
*/
module Cal.Core.String;
import Cal.Core.Prelude using
    typeClass = Eq, Inputable, Outputable;
    typeConstructor = Boolean, Char, Int, Maybe, Ordering, String;
    dataConstructor = False, True, Nothing, Just, LT, EQ, GT;
    function =
        assert, compare, concat, concatMap, eager, empty, isEmpty, not;
;
import Cal.Collections.List;
import Cal.Core.Char;
import Cal.Utilities.Locale using
    typeConstructor = Locale;
;
//we want to maintain the generality and functional independence of the String module.
//Please do not add more imports.

/**
 * Converts the string to a list of characters.
 *
 * @arg stringValue the string to be converted.
 * @return a list of characters in the string.
 */
toList :: String -> [Char];
public toList !stringValue =
    let
        stringToCharHelper :: String -> Int -> Int -> [Char];
        stringToCharHelper !s !i !len =
            if i >= len then
                []
            else
                (subscript s i) : (stringToCharHelper s (i + 1) len);
    in
        stringToCharHelper stringValue 0 (length stringValue);
```

The CAL Editor has many of the features found in the Java Editor, such as bookmarks, copy/cut/paste, undo/redo and local file history.

As you can see in the screen shot above, the CAL editor provides syntax coloring to identify comments, Caldoc comments, keywords, literals and type names. The font and color applied to these elements can be configured in the preference page at CAL > Editor > Syntax Coloring.

The editor automatically indents each line appropriately as you type and highlights matching brackets.

If the CAL Builder is enabled, the module is rebuilt as soon as you save the file. Compilation errors are indicated by a marker in the left margin.

The context menu provides the following commands:

Open Declaration (**F3** or Control+Left Mouse)

Goes to the declaration of the selected identifier. This work for foreign function as well so you can traverse to the Java code.

Source > Toggle Comment (**Ctrl-7** or **Ctrl-/**)

Adds // to the beginning of each selected line – or removes them if present

Source > Add Block Comment (**Ctrl-Shift-/**)

Surrounds the selected text with a block comment, delimited by `/*` and `*/`

Source > Remove Block Comment (**Ctrl-Shift-**)

Removes a block comment surrounding the selected text

Source > Correct Indentation (**Ctrl-I**)

Adjusts the indentation of the selected lines to match the CAL standard

Source > Organize Imports (**Ctrl-Shift-O**)

Adds or removes imported CAL modules, types, functions etc. as needed

Source > **Insert Type Declarations**

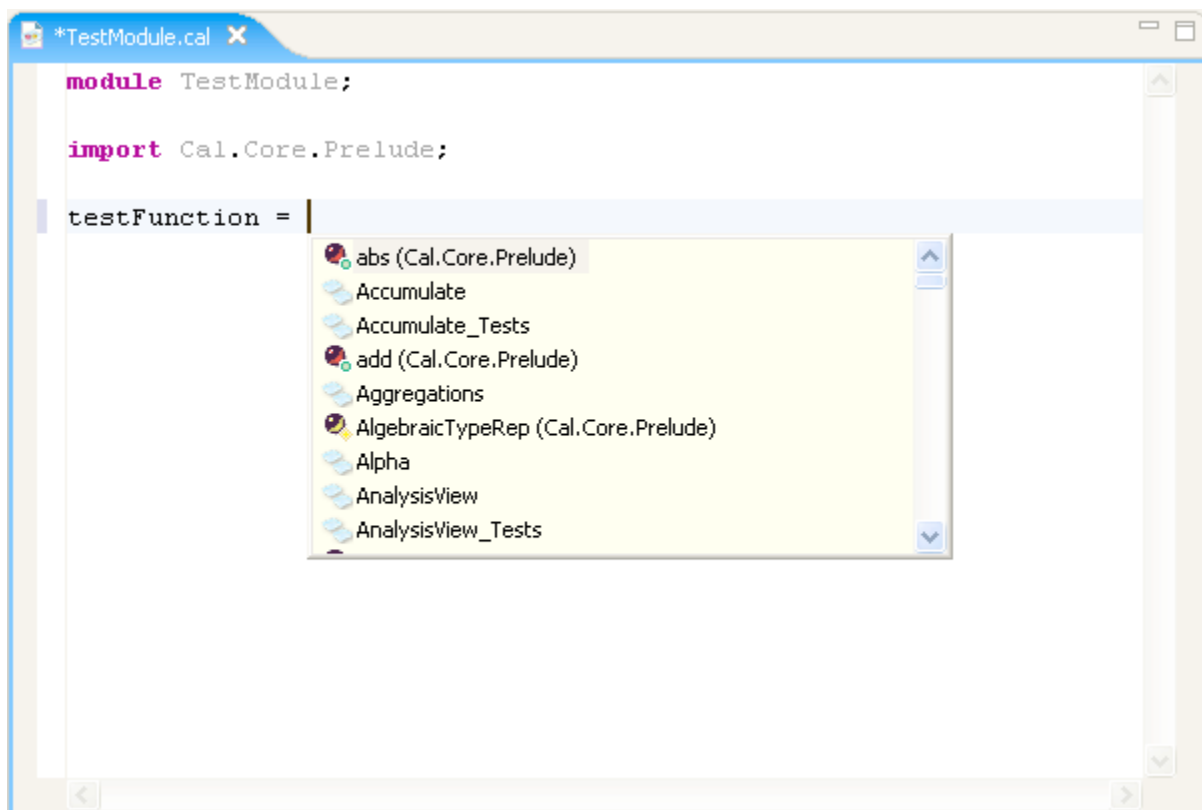
Insert type declarations for the functions in the selected range if they are missing type declarations.

Refactor > Rename... (**Alt-Shift-R**)

Renames a CAL entity, updating the definition and all references

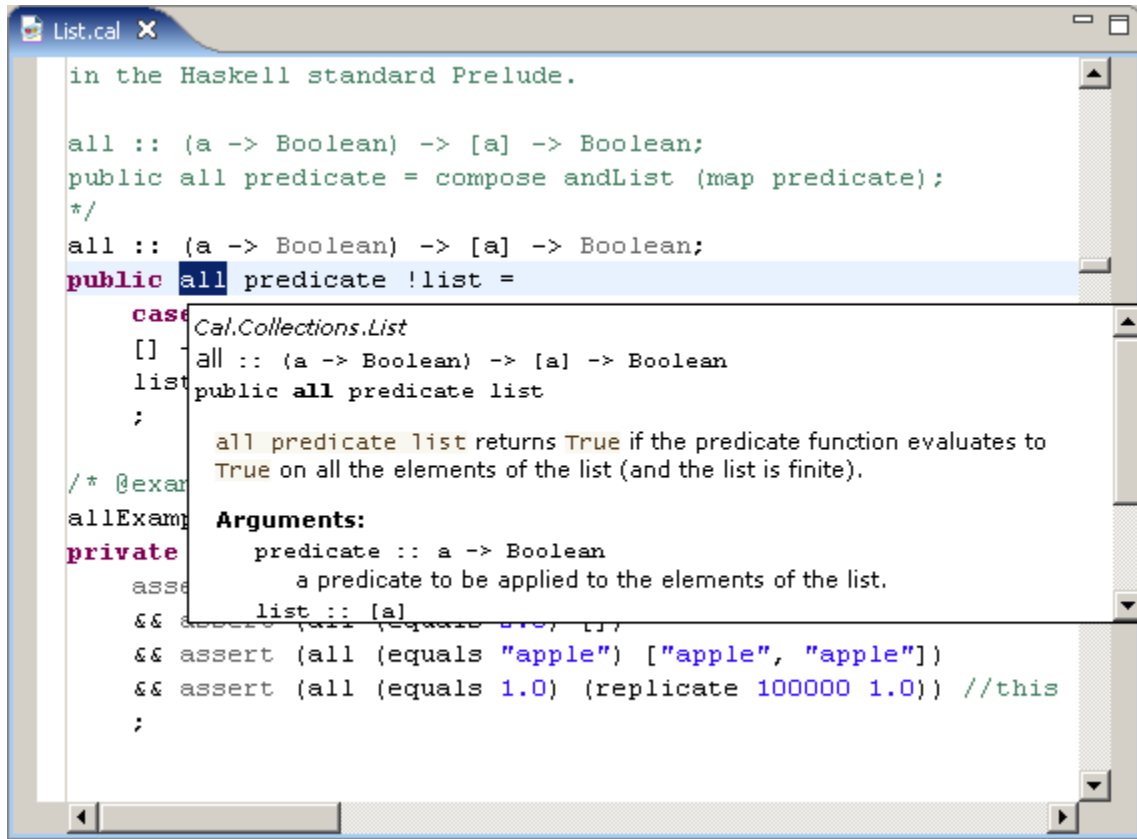
Auto Completion

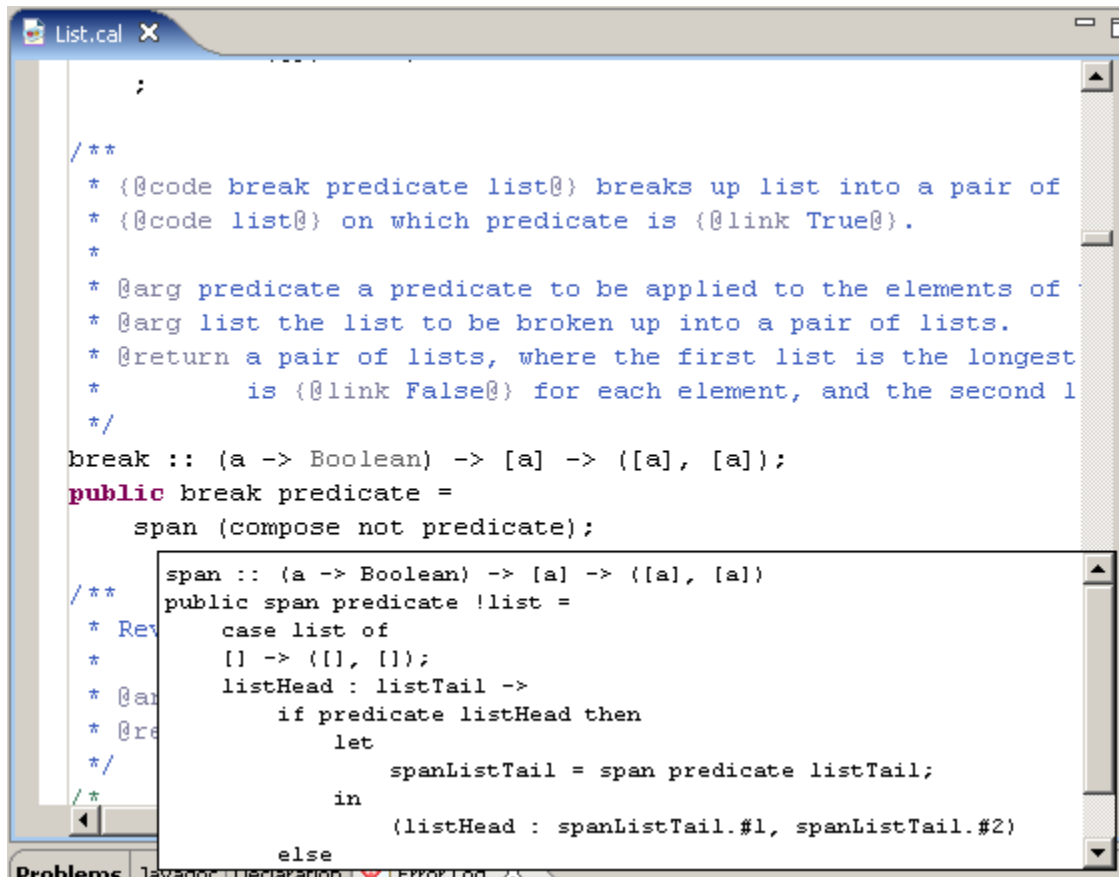
The CAL Editor supports auto-completion of statements. This is accessed via the control space command key as in the Java editor. Templates are also available for standard CAL expression format. This is especially useful for foreign function definitions. There is a property page for controlling the auto-complete behaviour as well.



Hovers

The CAL doc of a symbol is displayed during hovers over a symbol. If the control of shift key is held down then the source code of the symbol is displayed along with type information. Pressing F2 will cause the display to become sticky.



The image shows a screenshot of the CAL Editor. The main window, titled 'List.cal', contains the following code:

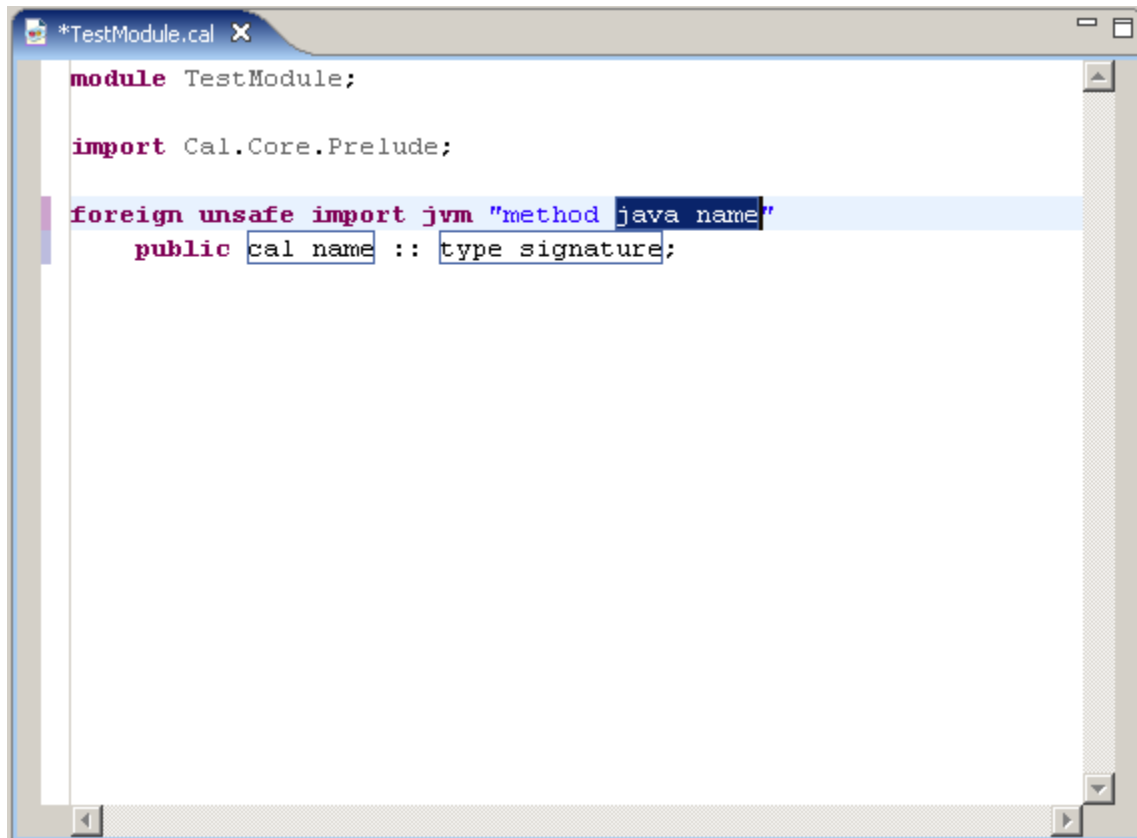
```
;  
  
/**  
 * {@code break predicate list} breaks up list into a pair of  
 * {@code list} on which predicate is {@link True}.  
 *  
 * @arg predicate a predicate to be applied to the elements of  
 * @arg list the list to be broken up into a pair of lists.  
 * @return a pair of lists, where the first list is the longest  
 *         is {@link False} for each element, and the second l  
 */  
break :: (a -> Boolean) -> [a] -> ([a], [a]);  
public break predicate =  
    span (compose not predicate);  
  
/**  
 * Rev  
 *  
 * @arg  
 * @re  
 */  
/*  
/*
```

A tooltip is displayed over the `span` function, showing its definition:

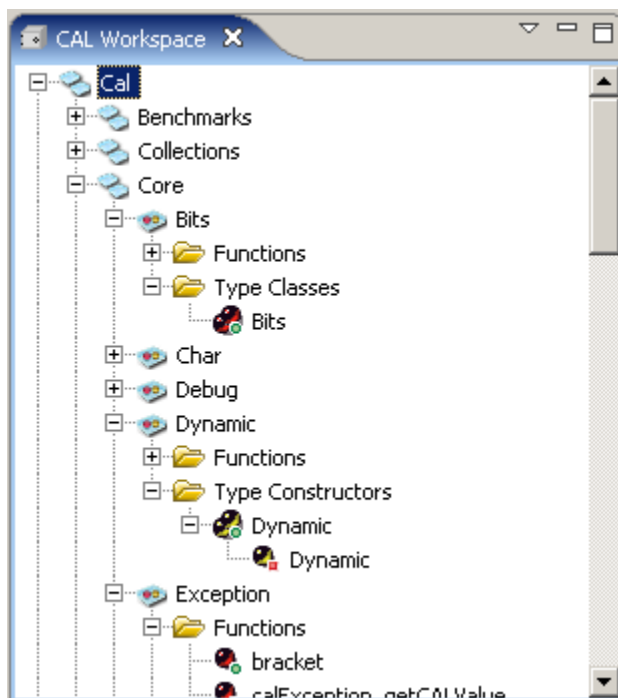
```
span :: (a -> Boolean) -> [a] -> ([a], [a])  
public span predicate !list =  
    case list of  
    [] -> ([], []);  
    listHead : listTail ->  
        if predicate listHead then  
            let  
                spanListTail = span predicate listTail;  
            in  
                (listHead : spanListTail.#1, spanListTail.#2)  
        else
```

Templates

The CAL Editor has a number of templates defined that support standard cal expressions. They are accessed by invoking auto-complete. These can be quite useful for setting up standard expressions. There is a property page for defining new templates.



2. The CAL Workspace View



This view shows all the CAL modules in all projects in the Eclipse workspace, arranged in a tree. Under each module there are nodes for each type, constructor, type class and function found in the module. The CAL element icons are marked with indicators that correspond to the scope of the entity. A green dot marks public objects, a yellow diamond marks protected objects and a red square marks private objects.

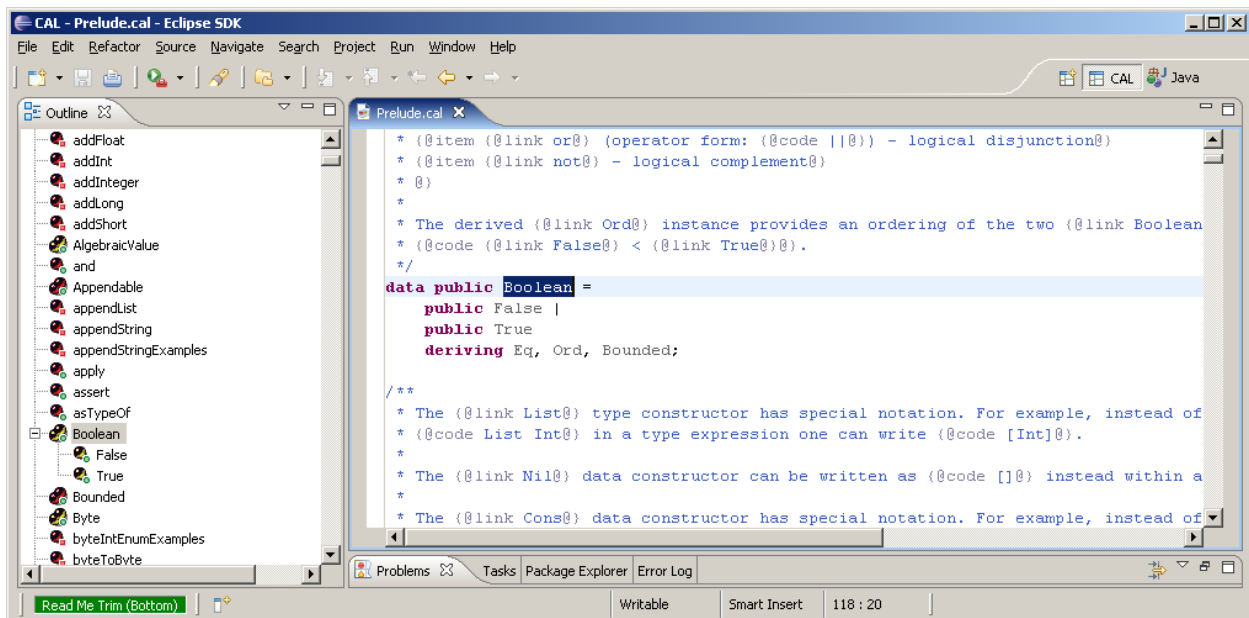
When you double-click on a CAL entity in the view, the CAL Editor opens the corresponding module and selects the entity.

You can customize the display of the tree using commands on the menu found in the upper-right area of the view. These commands are also on the context menu that is shown when you right-click on a node.

The context menu also has commands that allow you to drill into part of the tree. For example, you can drill into a module, causing the tree to display just the entities that are found in that module. You can drill again into the functions of the module. There are also commands to back up a step in the drill sequence, and to return to the original display of all modules.

If you right-click on a node that represents an applicable CAL construct (that is, a module, type, constructor, type class or function), the context menu will have a command that will open the CAL Metadata Editor on the selected construct. The metadata editor is described below.

3. The Outline View



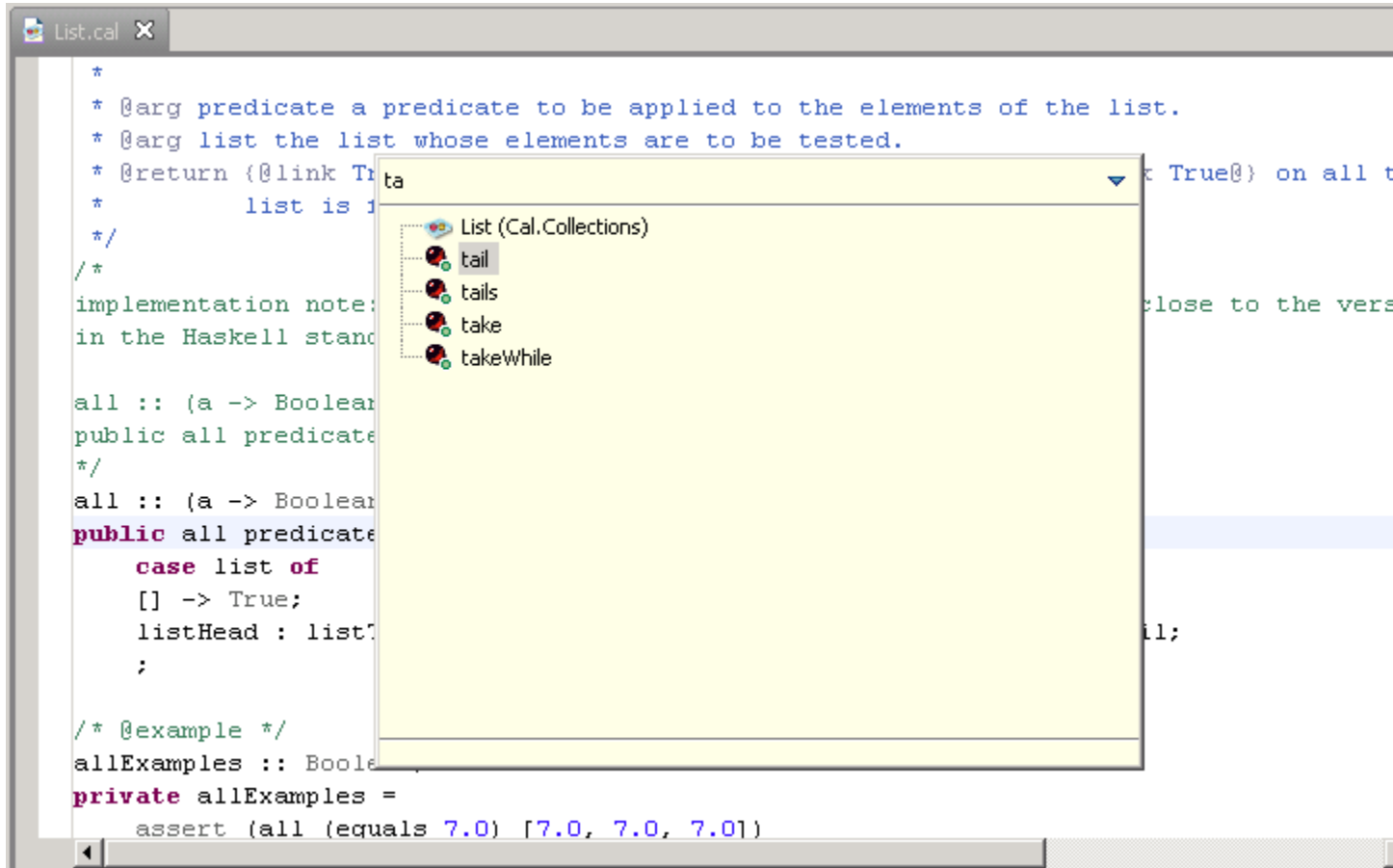
When the CAL editor is active, the standard Outline view shows a tree of the types, constructors, type classes and functions in the CAL module being edited.

When you double-click on an entity in the view, the CAL Editor opens the corresponding module and selects the entity.

You can choose to keep the Outline synchronized with the editor, so that as you change the selection in the editor, the corresponding node in the Outline is selected.

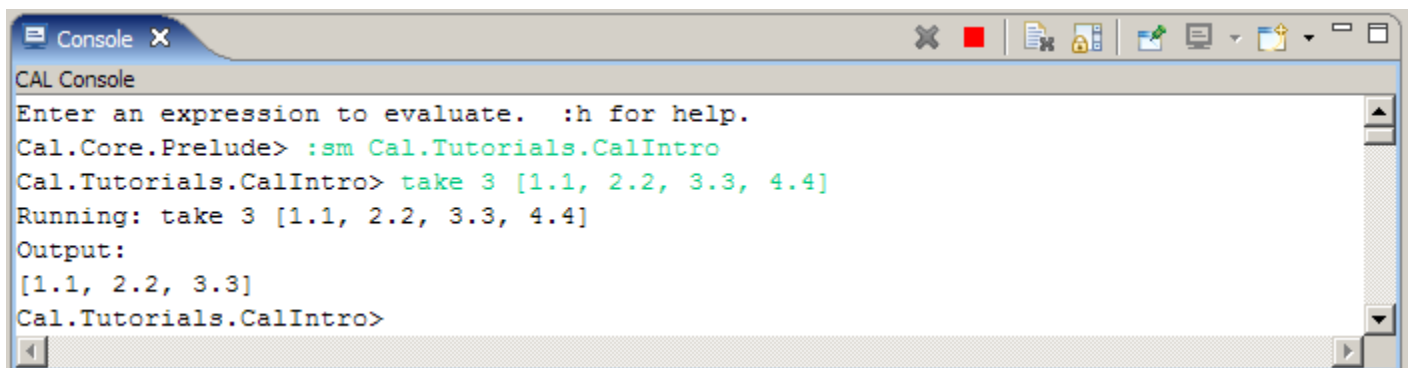
If you right-click on a node that represents an applicable CAL construct (that is, a module, type, constructor, type class or function), the context menu will have a command that will open the CAL Metadata Editor on the selected construct. The metadata editor is described below.

4. The Quick Outline View



The Quick Outline view shows a tree of the types, constructors, type classes and functions in the CAL module being edited. It can be invoked by pressing Control-O or by using the editor context menu. Entries in the list can be filtered by typing in pattern strings in the filter box. By selecting an entity the current position in the editor will be moved to that entity in the editor.

5. The CAL Console



The CAL console allows CAL expressions to be entered and evaluated with respect to the CAL modules in the Eclipse workspace.

The console can be opened by selecting **Run** → **Open CAL Console** from the Eclipse menu. The module with respect to which expressions are evaluated is displayed with the console prompt. To evaluate a CAL expression, simply enter the expression at the prompt. To change the current module, enter **:sm** *New_Module_Name*.

The console is also shown when interacting with elements in the CAL Workspace or outline views. A runnable functional agent can be evaluated in the console by selecting "Run in Console" in its context menu. Similarly, a given module can be set to be the active module by selecting "Open in Console" in its context menu.

Expression evaluation can be terminated at any time by clicking the stop button on the toolbar.

6. The CAL Metadata Editor

The screenshot shows the 'Function List.filter' metadata editor. The window has a title bar with 'List.filter' and standard window controls. The main content area is divided into several sections:

- Basic Properties:** Contains fields for Display Name, Short Description, Long Description (with a text area containing 'Applies a function to each element of a list and returns a new list containing the elements for which the function returns True.'), Author (set to 'Business Objects'), Version, Preferred (radio buttons for Yes and No, with No selected), and Expert (radio buttons for Yes and No, with No selected).
- Related Features:** A list box containing 'Cal' and 'UserGuide' with '+' and '-' icons. To the right are 'Up' and 'Down' buttons.
- Gem Properties:** A section header.
- Gem Arguments and Return Value:** Contains three entries:
 - keepIfTrueFunction :: a -> Boolean:** Display Name: keepIfTrueFunction, Short Description: This predicate returns True for items that should be kept, and False for items that should be dropped. A 'More...' button is to the right.
 - list :: [a]:** Display Name: list, Short Description: A list. A 'More...' button is to the right.
 - result :: [a]:** Short Description: (empty field).
- Usage Examples:** A section header.
- Custom Properties:** A section header.

The bottom of the window features a 'CAL Workspace' tab and a 'Problems' list. The Problems list shows three items: 'endsWithExamples', 'filter', and 'filterIndexed', each with a red error icon.

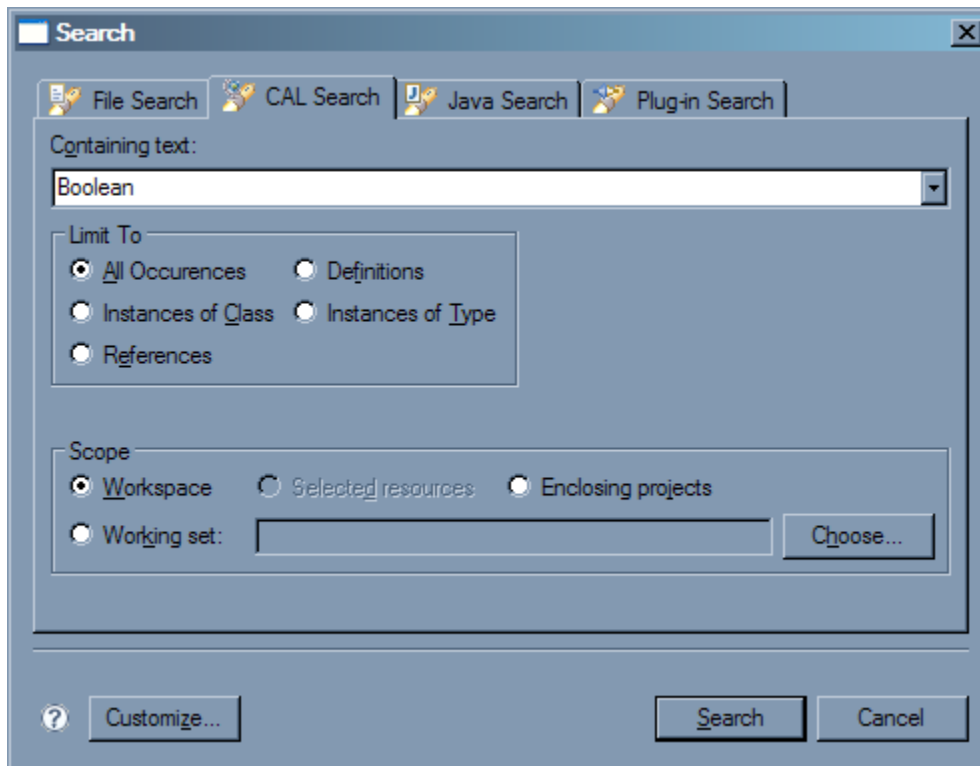
Many modules and entities in a CAL workspace have metadata associated with them. The metadata for these consists of a set of named properties. There are several pre-defined properties, such as Display Name, Short Description, Author and Related Features. It is possible to create a custom property by specifying a name and a string value.

Metadata is used by the Gem Cutter to enhance the display of gems on the Table Top. It is also used by some programs to annotate CAL entities in a manner specific to the program. For example, the BAM sample included in the Open Quark distribution uses metadata to tag trigger, metric and action gems.

The CAL Eclipse Plug-in includes a metadata editor that can be used to view or edit metadata. You can open the editor from the context menu in the CAL Workspace view or in the Outline view when a CAL file is being edited.

The editor is very similar to the Properties Browser in the Gem Cutter. The Gem Cutter Manual has a detailed description of the browser.

7. CAL Search



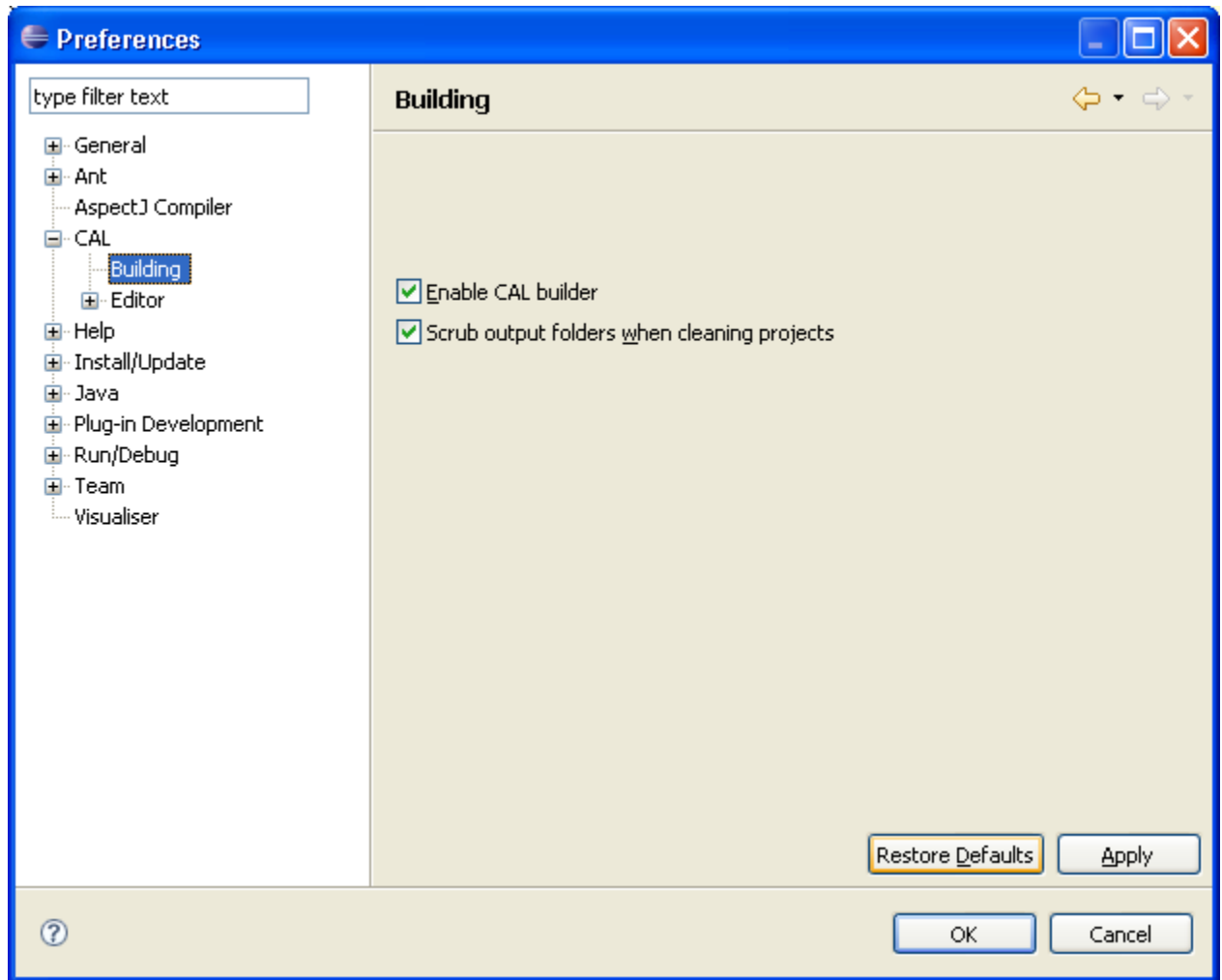
Using CAL Search, you can find different kinds of occurrences of CAL entities in the modules in the Eclipse workspace. The results are displayed in the Search view.

8. Preferences Pages

The CAL Eclipse Plug-in contributes several pages to the Preferences dialog:

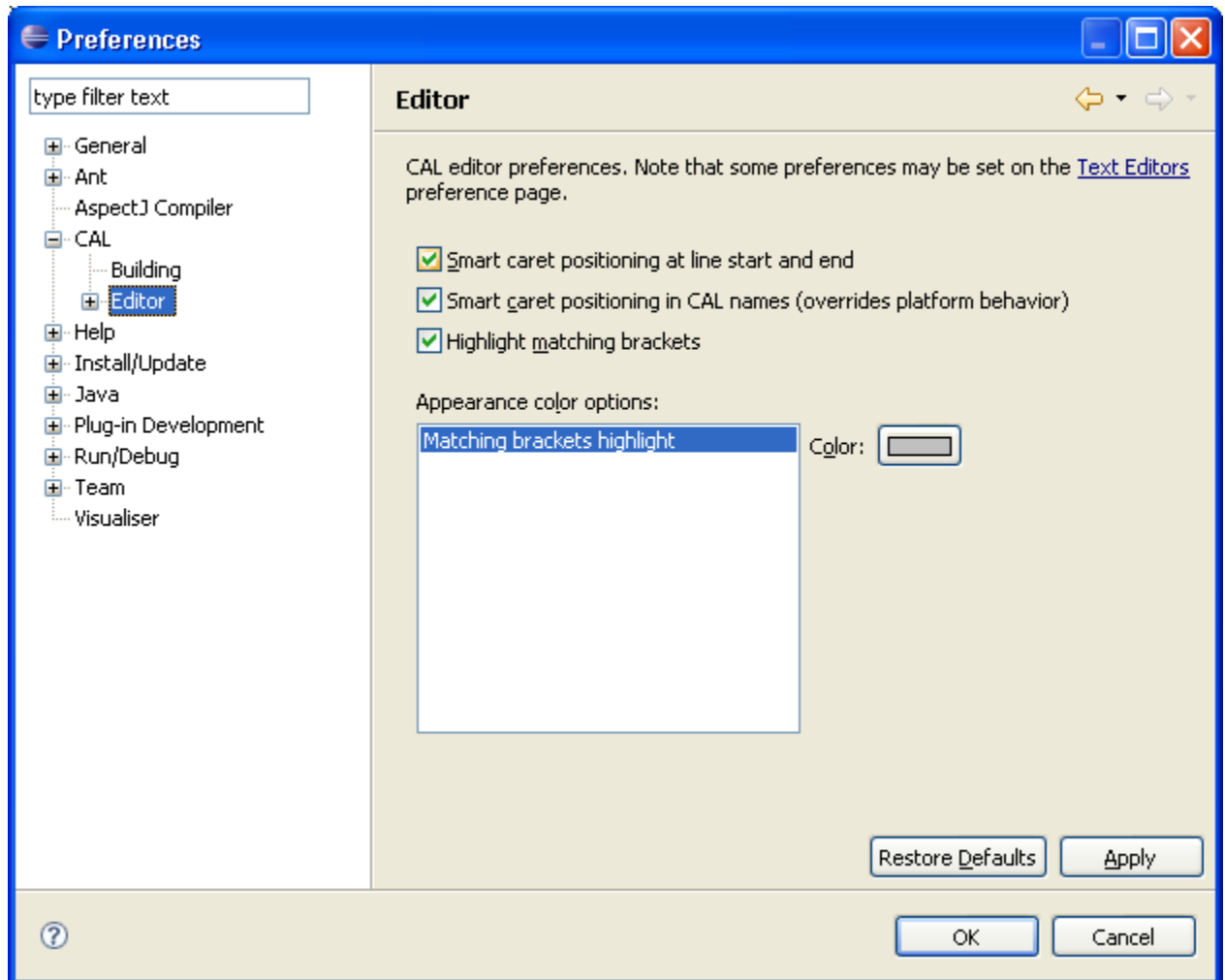
- CAL > Building

- Enable the CAL builder and control clean-up of output folders

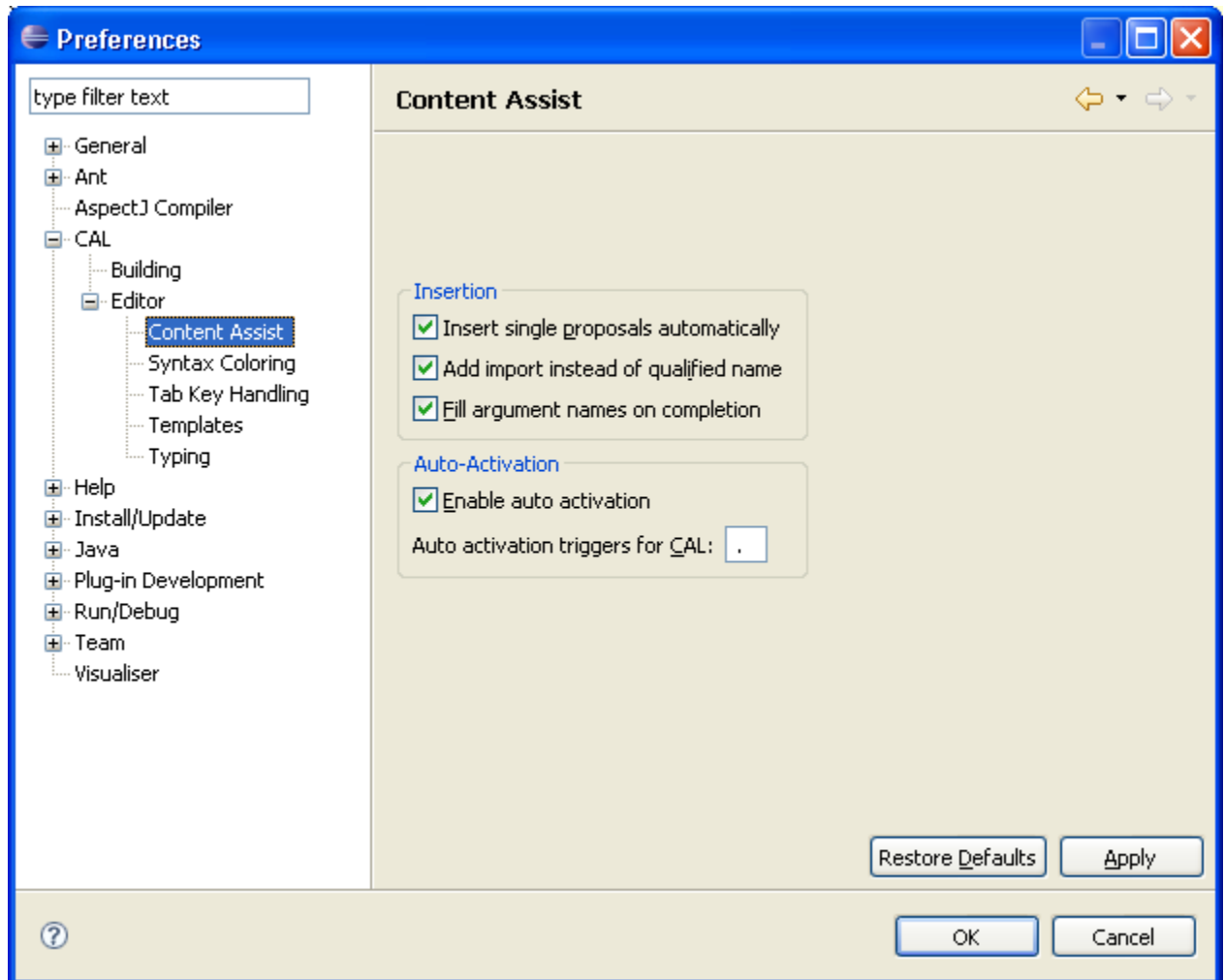


CAL > Editor

Control smart caret positioning and highlighting of brackets

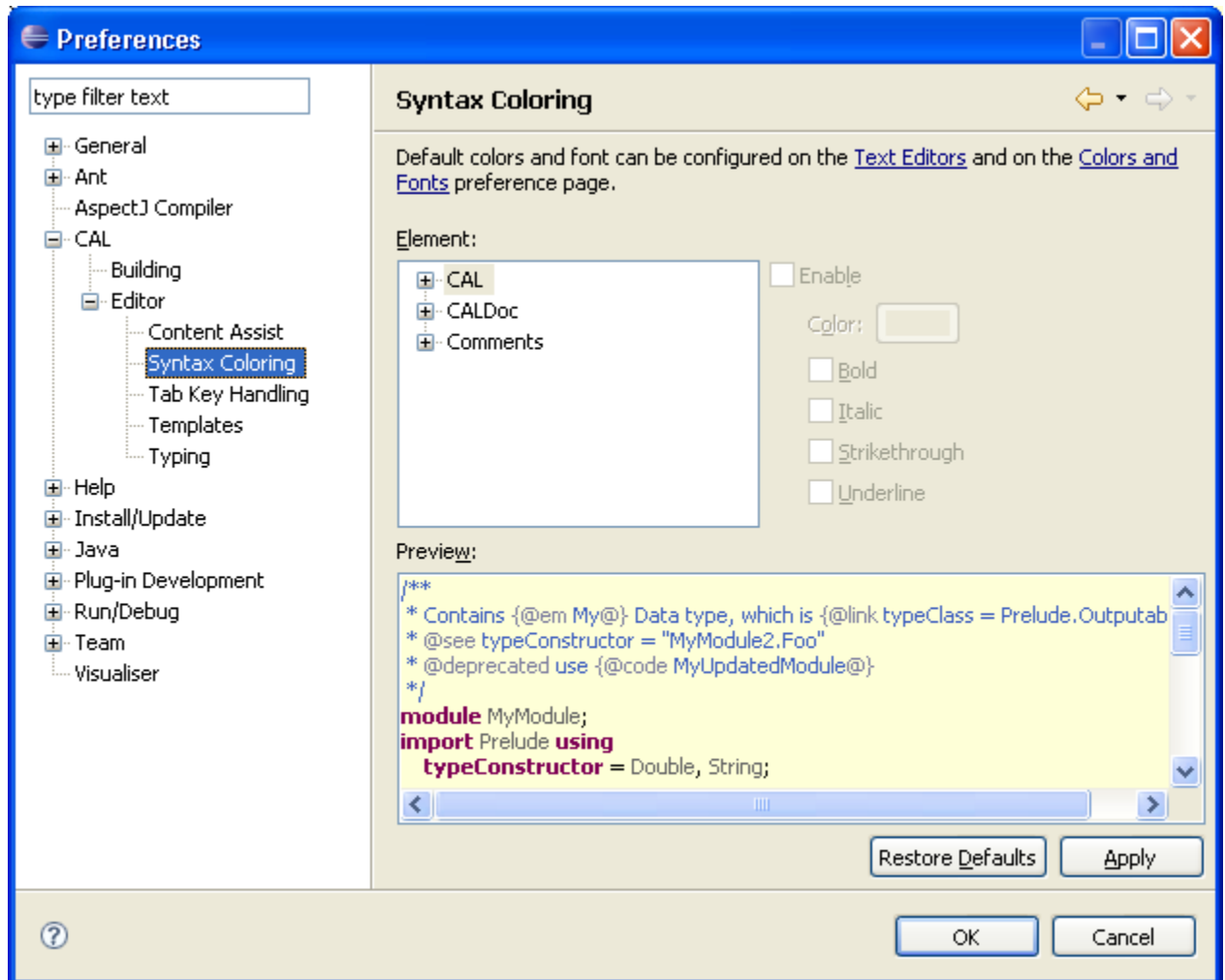


CAL > Editor > Content Assist
Control CAL Editor content assist preferences



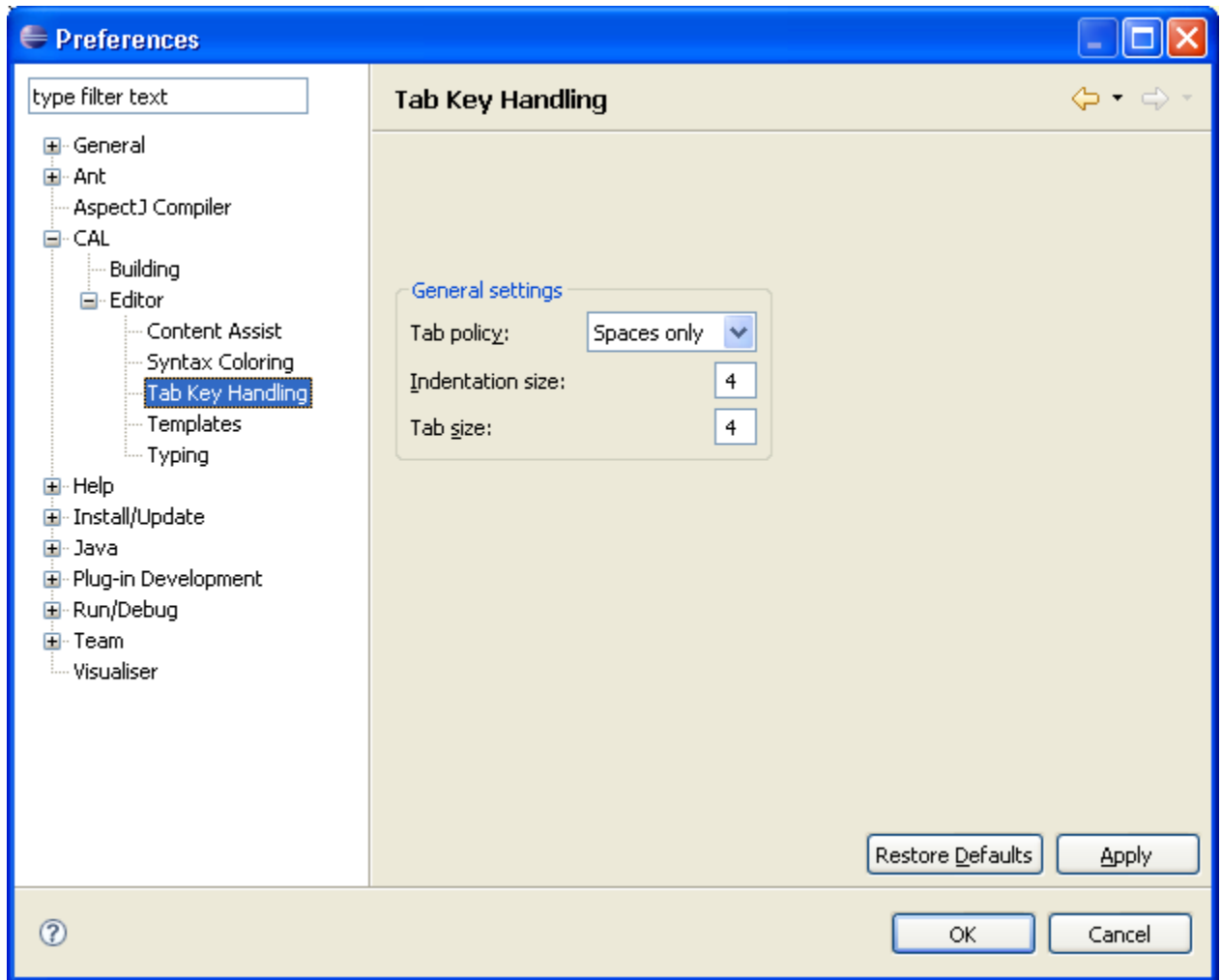
CAL > Editor > Syntax Coloring

Control fonts and colors used for CAL language elements



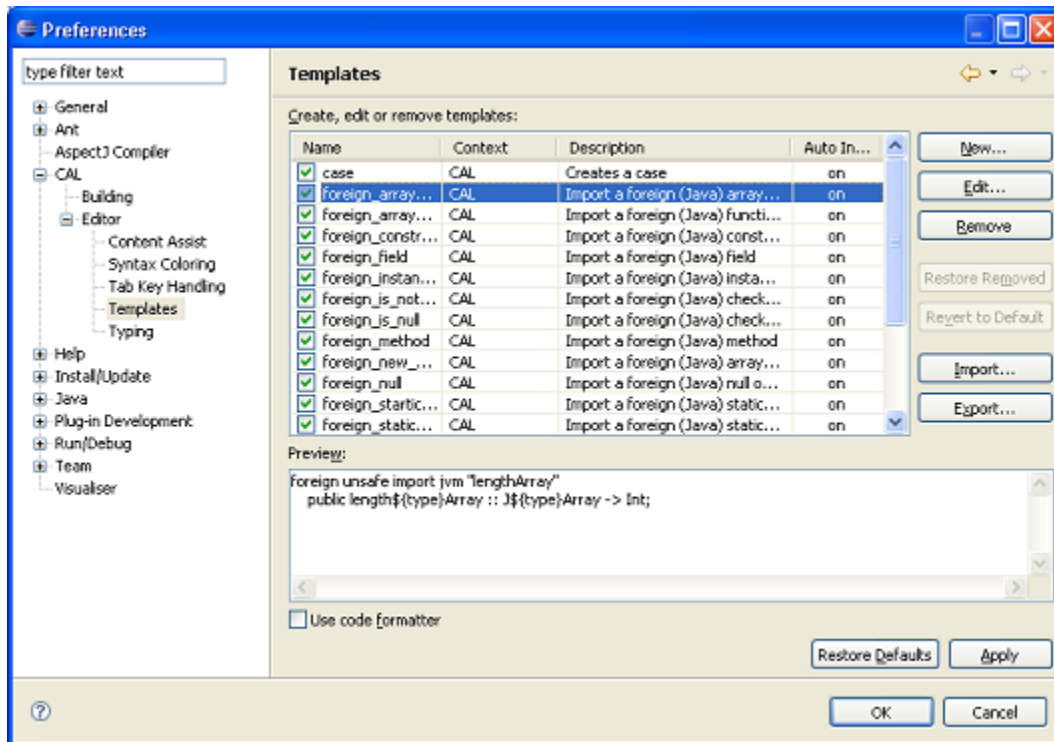
CAL > Editor > Tab Key Handling

Choose whether to insert spaces or a TAB character when the Tab key is pressed



CAL > Editor > Templates

Create edit or remove code templates. This preference page is similar to the Java Template Preference Page (Java > Editor > Templates). Here you can create and edit code templates that are available in the CAL Editor when content assist is used.



CAL > Editor > Typing

Control options for automatically closing strings, bracket pairs and comments, etc.

