Business Activity Monitor Overview

Contributor: Magnus Byne
Last modified: August 24, 2007

Contents

Overview

This document describes the Business Activity Monitor (BAM) demo application, which is supplied with Quark. The BAM demo is intended to go beyond the basic sample code and show how a non-trivial program can be developed using Java and Quark.

Document layout

This document is divided into the following sections:

User Overview describes what the BAM application does, and how to use it.

Design Overview provides a brief overview of the design of the BAM application.

Gem Graph describes the gem graphs that are constructed by the BAM application to process messages.

Developing new gems for BAM describes how to extend BAM with new actions, triggers and metrics.

Call for feedback

This document aims to be as complete and easy-to-use as possible. Any feedback you might have to help improve it would be 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).

User Overview

The Business Activity Monitor is an application for monitoring messages sources and performing actions whenever specific conditions are met.

The user associates a number of triggers and actions with each message source. If all the triggers evaluate to true for a particular message, all the associated actions are performed for that message. The triggers may depend on constant values, message properties, or metrics which are computed over the stream of messages.

Starting BAM

To start BAM run `BAMSample.bat` or `BAMSample.sh` (depending on your operating system) from the `samples/bam` directory.

Using BAM

The BAM application has a complete user interface implemented using the Java Swing API. The main BAM window is shown below in Figure 1. It has two main tabs, an editor tab and a log tab. The editor tab has three sections. The leftmost list shows the configured message sources. When a message is selected in this list the two lists on the right show the corresponding triggers and actions. The log tab is used to display log messages when the application is running.

In order to quickly set up message sources, triggers and actions for demonstration purposes, the Test menu includes a 'Make Test Document' item, which populates the application with a testing setup that is ready to run.

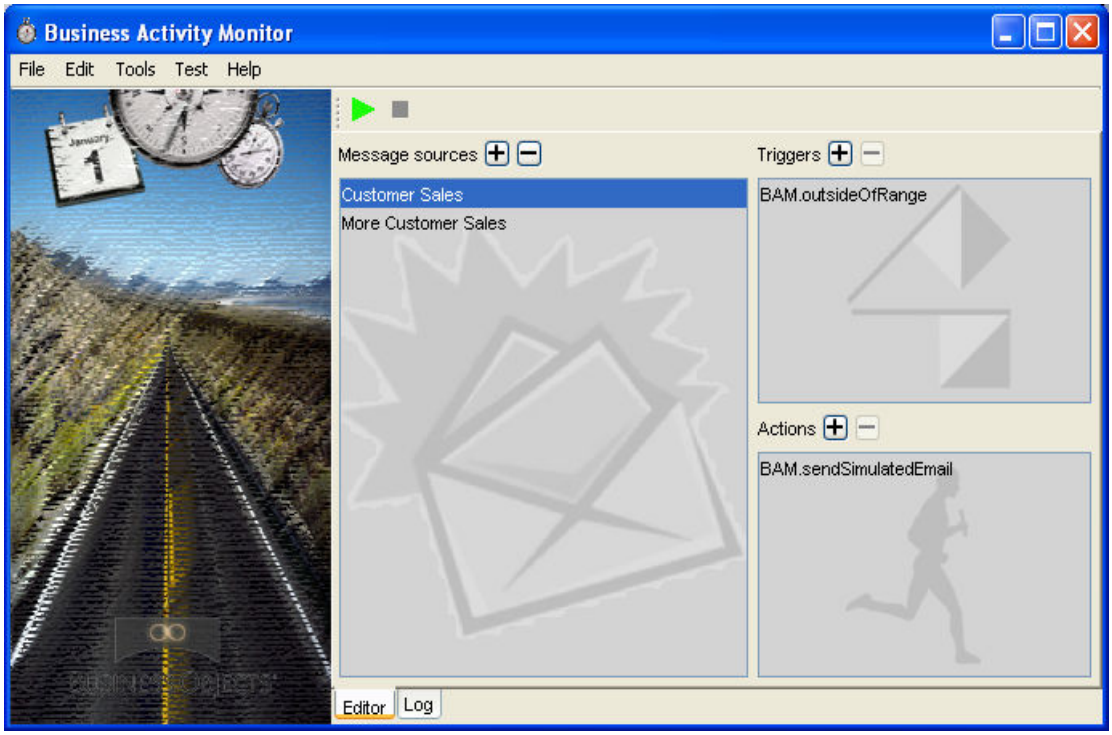


Figure 1 Main Window.

Configuring a message source

New message sources can be added using the Edit -> Add Message Source menu item. This opens a wizard for creating a new message source. BAM supports file based message sources, which read messages from a text file, and random message sources which create random messages. For real applications BAM would probably have to be extended to support the JMS API.

Adding Triggers

Triggers can be added using the Edit -> Add Trigger menu item. This opens the trigger wizard, which allows the users to select the trigger to use from a list of appropriate gems found in the application's Quark workspace. Once the trigger has been added the user must bind all of the trigger's inputs. The wizard pane for this is shown below:

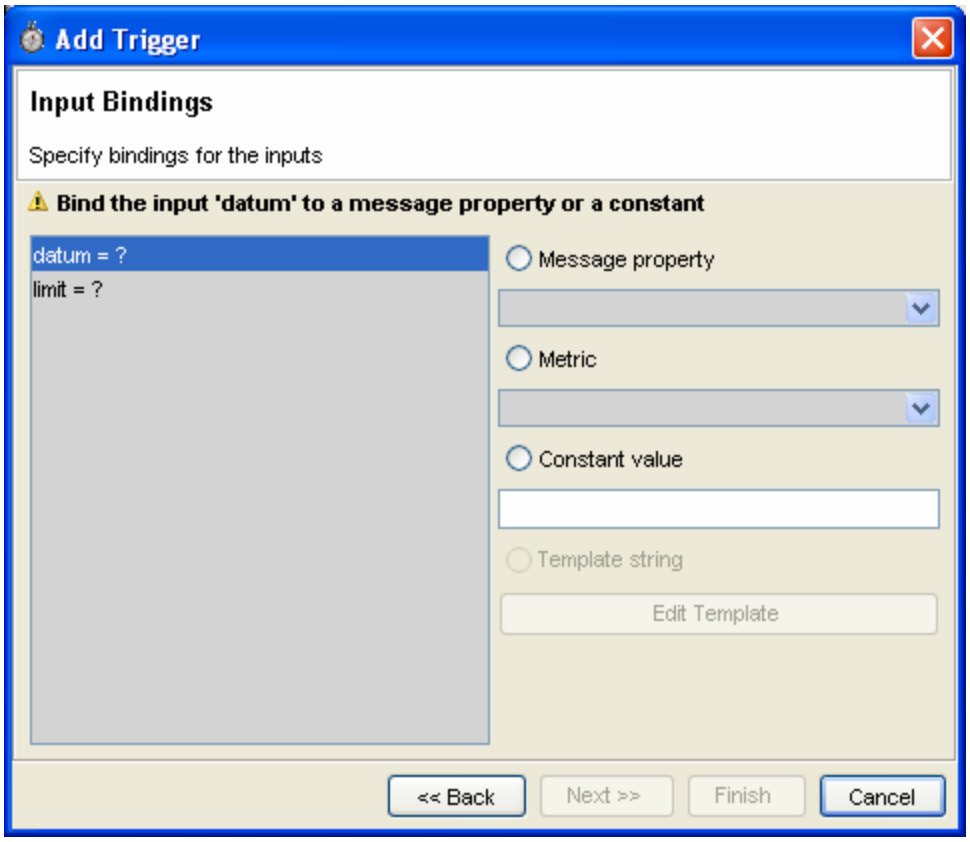


Figure 2 Trigger Input Bindings

All of the trigger inputs are shown in the leftmost list. Unbound inputs are indicated by the '?' symbol. Each input can be bound to a message property, a metric or a constant value. The drop down list of message properties includes all the message properties that have a compatible type. The drop down list of metrics includes all the possible combinations of the available metrics and message properties that result in a compatible type. The constant value option allows the user to enter a fixed value of the appropriate type. Once all the trigger inputs have been bound, the new trigger can be added by clicking Finish. Existing Triggers can be edited by right clicking on them in the trigger list and choosing edit.

Adding Actions

New actions can be added using the Edit -> Add Action menu item. Actions are configured in a very similar way to Triggers. The Action wizard first allows the users to select the action from a list of available actions, and then configure the bindings for each of the action's inputs. String inputs can be bound to template strings, which are useful for creating formatted messages, for example for the contents of an email message. Clicking the edit template string opens the template editor, shown below:

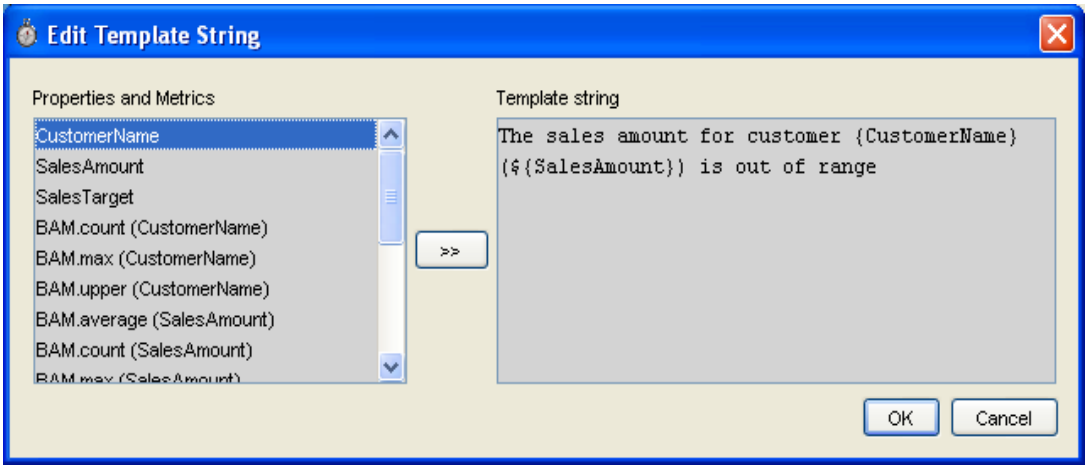


Figure 3 Edit template window.

The list on the left shows the available message properties and metrics which can be formatted as strings. They can be inserted into the template string as placeholders using the '>>' button. In the example shown the CustomerName and SalesAmount properties have been inserted. They are denoted with the curly brackets. Whenever the action is performed these placeholders will be replaced by the actual values from the current message before the template string is supplied to the action input.

Starting Message Processing

Once the message sources, triggers and actions have been configured, the message processing can be started and stopped using the play and stop buttons. When the application is running various logging messages (including the actual messages and actions performed) are written to the log tab of the main window. An example is shown below:

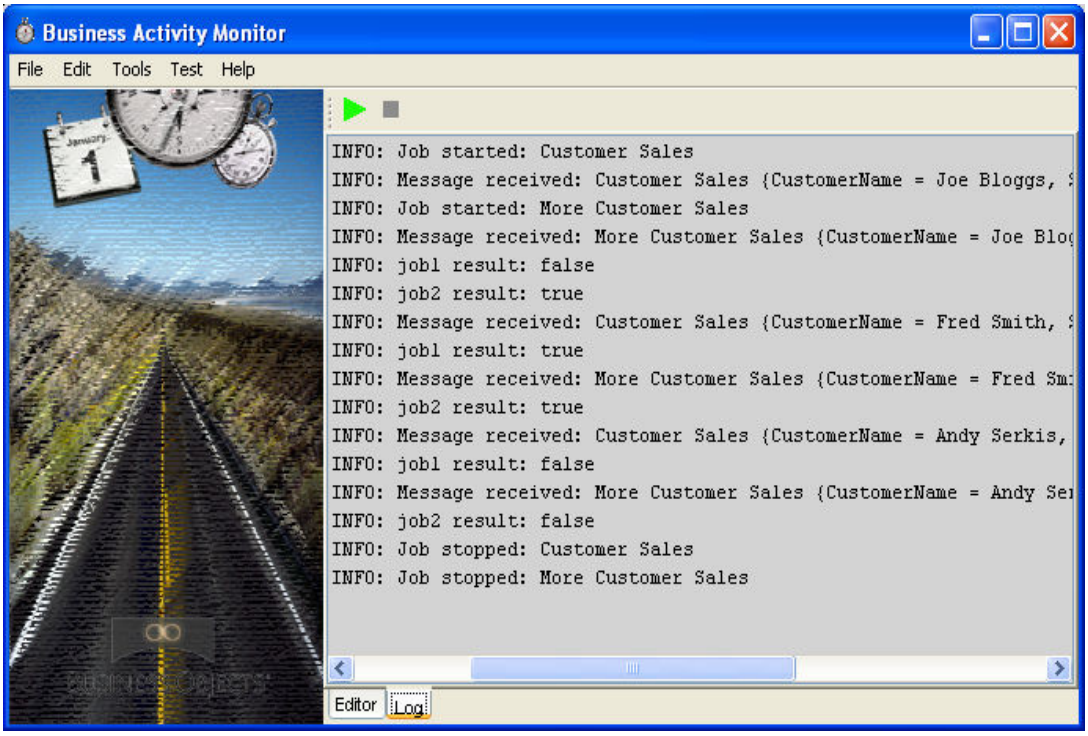


Figure 4 Example Log Output

Manage Gems

The manage gems dialog is available from the Tools -> Manage Gems menu item. The dialog is used to control which gems are shown as actions, triggers and metrics. It shows all the gems in the Quark workspace that have the appropriate signature and allows the user to select them for inclusion in the trigger and action wizards.

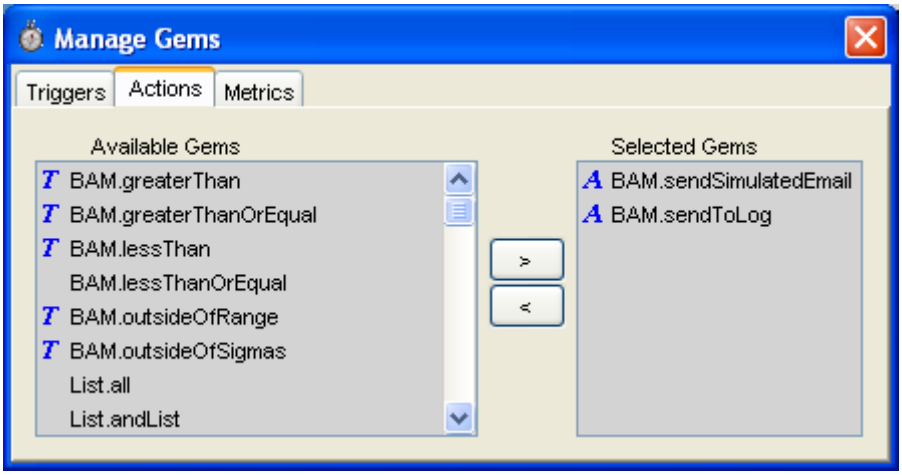


Figure 5 Manage Gems Dialog

Design Overview

The BAM demo application is implemented using Java and Quark. Java is used for the user interaction and interfacing with the message queues. Quark is used to implement the message processing logic.

Triggers, Metrics and Actions

The triggers, metrics and actions in BAM are implemented in CAL. They are all defined in `samples/BAM/BAM.cal`.

Each trigger function takes one or more inputs and returns a `Boolean` result. Using the BAM interface, each input can be bound to a constant value, a value from the message, or a metric computed from the stream of messages.

Here is an example trigger function that tests whether one value is greater than another:

```
greaterThan :: Double -> Double -> Boolean;
public greaterThan datum limit = datum > limit;
```

An action function also takes one or more inputs and returns a `Boolean` result. Here the result indicates whether the action succeeded. As with trigger functions, the inputs of an action function can be bound to a constant value, a value from the message, a metric, or additionally a template string.

A template string includes slots into which values from the current message or metrics are placed. Template strings are typically used to generate formatted messages for presentation to the user.

Here is an action function logs a message:

```
sendToLog :: String -> Boolean;
public sendToLog message = csSendToLog message;
```

In BAM the action gems use methods that are implemented in Java (see the `CalSupport` class).

The metric functions in BAM compute running statistics. They take a list representing the values of a single message property over the complete sequence of messages, and output a list representing the value of the metric at each point in the message sequence. The i^{th} element in the output list represents the value of the metric at the i^{th} message.

Here is an example metric function that computes the running maximum of a message property:

```
max :: Ord a => [a] -> [a];
public max values = List.accumulateLeft1Strict Prelude.max values;
```

Building the Message Processing Task Description

Using the Java based UI the user can select the triggers and actions to associate with each message source.

The Java code uses the Quark API to search the BAM workspace for trigger, metric and action gems. (See the Java classes: `ActionGemFilter`, `MetricGemFilter` and `TriggerGemFilter`.) This makes the application very extensible: to add a new trigger, metric or action, a developer just has to add an appropriate CAL function (either programmatically or using the GemCutter) to the BAM workspace and it will automatically appear in the Manage Gems dialog.

The Quark type checker is used to determine the possible bindings for each of the trigger and action inputs. For example, the average metric is defined to map from any type of number to a `double`. The CAL type checker is used to check if this mapping can be used between a particular message property and a particular action input. This is used to only display the possible bindings in the user interface.

The complete set of triggers, actions and their bindings are represented by a job description (see the `model.MonitorJobDescription` class). This consists of a triplet containing the message source description, a list of triggers and their bindings, and a list of actions and their bindings.

Message Processing

When the message processing is started the Quark API is used to build a gem graph corresponding to each job description. See the section Gem Graph for more details.

In the gem graph the message processing task is modeled as a mapping from an input list, containing message records, to an output list containing Boolean values indicating whether or not the actions are performed for the corresponding message.

A message buffer (see the Java class `MonitorJob.MessageBuffer`) acts as a bridge between a Java message source and the gem graph. The Java message source produces messages that are inserted into the message buffer. The message buffer unpacks the messages and formats them for the gem graph. The gem graph uses

the `Iterator` interface to consume messages from the message buffer. The Java application uses the `Iterator` interface to extract the Boolean results from the gem graph's output list (see the Java class `MonitorJob`).

Quark's lazy evaluation scheme and the `Iterator` interface allow elements to be read from the output list as soon as they can be computed. In the demo all of the metrics are defined in terms of the preceding messages, so that the actions can be performed for a message as soon as the message is available. However, without any changes to the existing code a new metric could be added which could be defined in terms of the previous and the next message (for example, consider a local maximum metric). This would simply introduce a lag of one message to the outcomes which required this new metric.

Gem Graph

The gem graph is constructed by the `GemGraphGenerator` class. The gem graph models the message processing task as a function mapping from for a list of messages to a list of Boolean results, each indicating whether or not actions were preformed for the corresponding message.

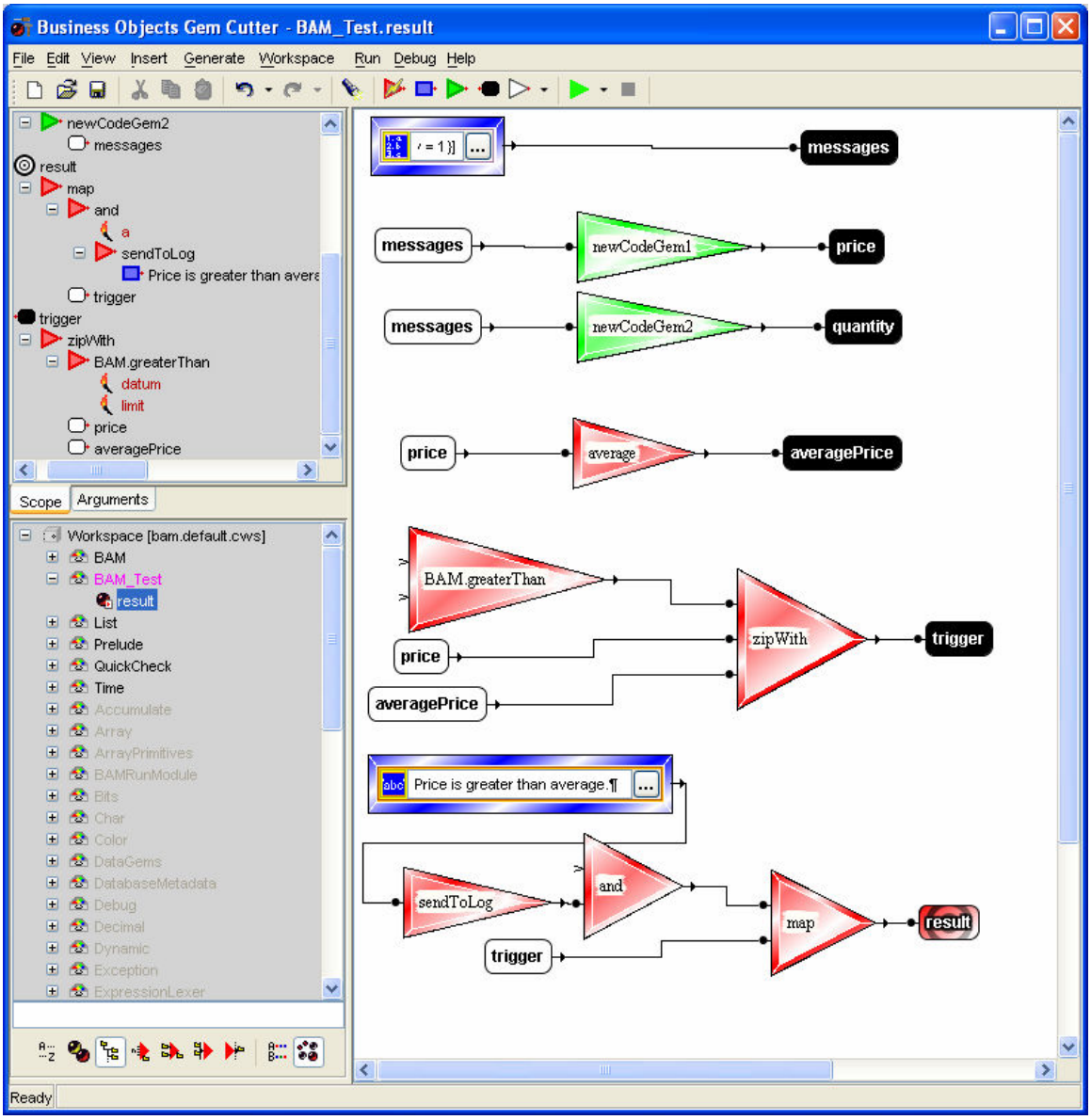
Unlike most gems, the action gems produce side affects, such as sending email and writing messages to logs. This is done using imported Java methods.

The gem graph that is constructed to process the messages has a number of components:

- A collector for the input list of message records.
- Collectors for lists of each of the message properties, which are computed using simple code gems.
- Collectors for each of the required metrics, which are computed by applying the metric gems to the message properties.
- A collector for the result of applying the trigger gems to their inputs (message properties, metrics and constants)
- A result collector which combines the action gems with the trigger result.

In the gem graph the Quark list `map`, `zipWith` and related functions are used to apply the trigger and action functions over the lists of message properties.

An example gem graph, constructed in the GemCutter, is shown below:



This gem graph is for messages that contain price and quantity properties. It computes the average metric for the price property. It applies the greaterThan trigger to the price property and the averagePrice metric. It performs the sendToLog action for messages when the trigger is true.

Developing new gems for BAM

This section describes how to extend BAM by adding new triggers, actions or metrics.

When the BAM application starts up, it loads the default BAM workspace. This is defined in the `samples/BAM/Workspace Declarations/bam.default.cws` file. The default workspace includes the `samples/bam/CAL/BAM.cal` file, which is where all of the BAM gems are defined. This is where new metric, trigger and action gems should be added.

New gems for BAM can be created in two ways, either by programming them directly in CAL, or by using the GemCutter.

Using the Gem Cutter:

Start the GemCutter (using `GemCutter.bat` or `GemCutter.sh` depending on your operating system). Once the gem cutter has started you must switch to the BAM workspace using the File -> Switch Workspace menu item, and then set the current module to BAM. Create new gems in the normal way – see the "Business Objects Gem Cutter Manual" for full details on how to do this. When a new gem has been created save it to the BAM module. When you restart BAM the new gem should appear in the 'Manage Gems' dialog, providing it has an appropriate type signature.

Programmatically:

Alternatively, new gems can be programmed in CAL directly by adding their definition to `BAM.cal`. It is possible to test the new gems using ICE (start using either `ICE.bat` or `ICE.sh` depending on your operating system) and switching to the BAM workspace and then to the BAM module:

```
:ldw bam.default.cws
:sm BAM
```

Once the new gems have been added and tested, restart BAM and include them using the 'Manage Gems' dialog.

