BabyIDE User Guide

Trygve Reenskaug,
trygver@ifi.uio.no

BabyIDE is an implementation of the Loke conceptual model as well as the Loke/Expert IDE. The DCI programming paradigm forms the foundation of the IDE, and it provides browsers for each of its Data, Context, and Interaction kinds of projections. The browsers appear as overlays in a shared window. The browsers carry their own state so that work in a browser can be suspended and resumed at will. The browsers are:

- **Data:** *The Data Class browser* is for working with personal Resource classes.
- **Context:** *The Context Class browser* is for editing a Context as a class. A Context class declares the provided interface that triggers the execution of a system operation. It also declares the private methods that bind Roles to objects during the execution of an operation.
- **Interaction:** *The Interaction browser* is for editing the Interaction diagram with its Roles and the links between them. It also supports creating and editing the RoleScripts that drive the collaboration of the Roles when they perform an operation.

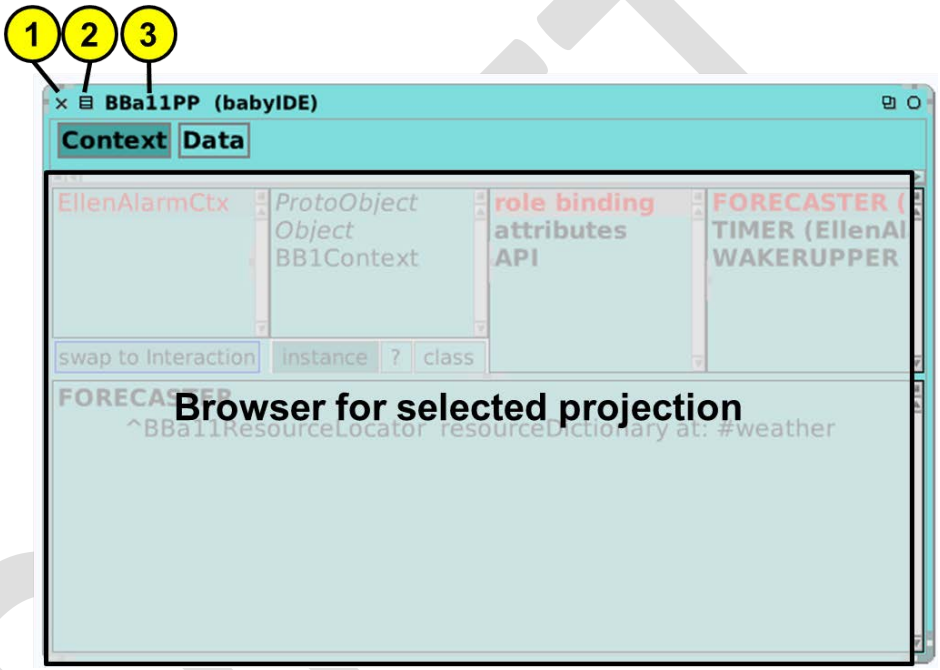
DRAFT

A. The shared window of all BabyIDE browsers

All BabyIDE browsers occupy the same area in the same window (Figure 1):

1. This button closes the window.
2. A menu button, its commands are
   - *Remove this app from the system.*
   - *Change the app that is handled by this IDE.*
   - *Add a new projection.* (Not implemented)
   - *Export this operation as a read-only HTML file*[1].
3. The application is *BBa11PP*, Ellen's Smart Alarm clock.
   (The obscure BBa11 name prefix is used to get around the lack of name spaces in Squeak. It confuses the code, and I have repressed it in this article whenever practicable).

Figure 1: BabyIDE heading
~~BB11PP-Window-annotated.png~~



[1] Example in http://folk.uio.no/trygver/assets/BB, a11PPEllen/readableVersion.html.

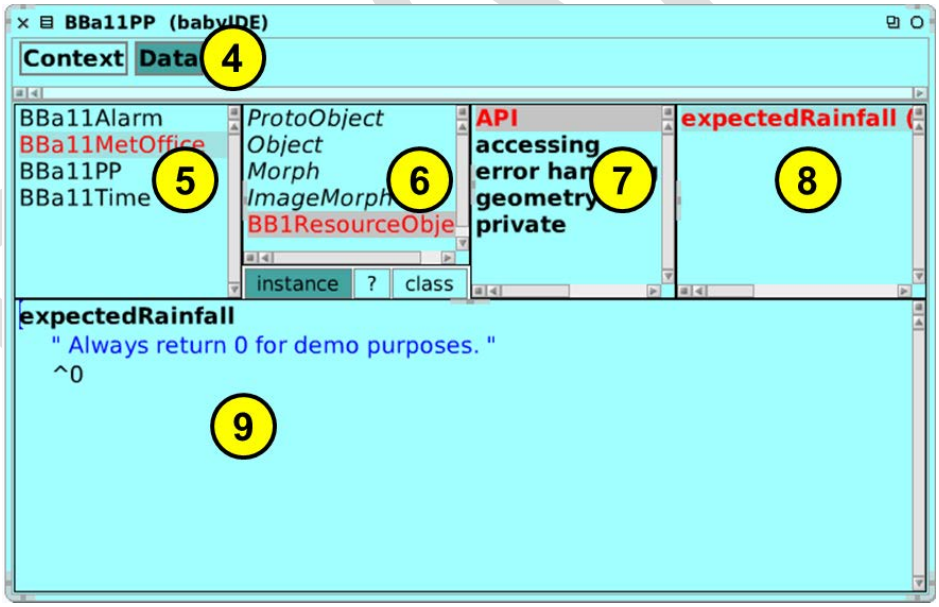
B. The Data Class browser

Any object with a globally unique identity and a RESTful message interface can play a Role in a Context.

The personal classes that are specific for the current operation are edited in the Data projection. The panes in the Data class browser are (Figure 2):

4. The selected projection is *Data*.
5. The class list shows the personal Data classes. *BBa11MetOffice* is selected. The classes are personal resource classes specifically declared for the current operation. Other classes and services are handled elsewhere.
6. A multi-select list shows the superclasses of the selected class and acts as a presentation filter.
7. A multi-select list shows the method categories of the selected class and any selected superclasses. It acts as a presentation filter and only shows the methods belonging to the selected categories. The *API* category is selected. By convention, the API methods form the provided interface of the Context.
8. A list of methods in the selected method categories. The *expectedRainfall*-method is selected.
9. A code pane for editing the source code of the selected method. Note the difference between a BabyIDE RoleScript and a Squeak method. The first is a property of a Role, and the second is a property of an object.

Figure 2: The Data class browser.
~~BB11PP-Data-annotated.png~~



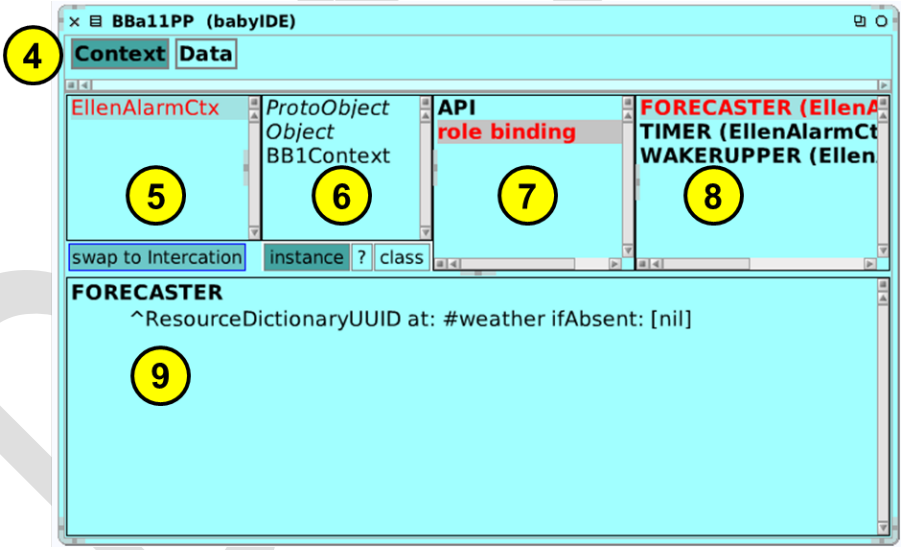
C. The Context Class browser

A Context instance (aka a *Maestro*) forms the environment for the execution of RoleScripts. A Context Class Browser is used to edit the properties of the Maestro.

Its panes are (Figure 3):

4. The projection is named *Context*.
5. A list of Contexts, *EllenAlarmCtx* is selected. A button toggles to the *Interaction* browser.
6. A multi-select presentation filter that shows the superclasses of the selected class.
7. A multi-select presentation filter that shows the method categories of the selected class and any selected superclasses. The *Role binding* category is selected. By convention, the *API* category is reserved for the provided message interface of this Context, i.e., the system operations implemented by this Context.
8. A list of methods in the selected method categories. There is one method for each Role in the *role binding* category; it binds the Role programmatically to a resource object. *FORECASTER* is selected. The role binding methods are always executed together as one atomic operation to ensure consistency.
9. A code pane for editing the source code of the selected method. This text pane that lets the programmer write code in Squeak's default language for methods.

Figure 3: The Context Class browser.
~~BB11PP-ContextClass-3-annotated.png~~



Note that an instance of a Context class (a Maestro) shall be a subclass of Resource *to enable it to play a Role in an outer context. The figure shows that this feature is not yet implemented: Currently,* *(BB1)Context* is a subclass of Object. instead of Resource.

D. The Interaction Browser

The Interaction browser (Figure 4) is where the programmer specifies how a BabyIDE realizes an operation with a network of collaborating objects. The code in this projection answers three critical questions for each operation:

- *What are the objects?* A Role identifies a participating object and is a placeholder that gives the object a name at compile time. The objects actual identity is determined at runtime.
- *How are they interlinked?* The programmer answers this question by defining a network of Roles in the Interaction Diagram.
- *What do they do?* The Programmer answers this question by augmenting selected Roles with *RoleScripts*. The BabyIDE runtime system creates the illusion that these scripts are parts of the Roleplaying objects at runtime. In reality, the Roleplaying objects never change.

The panes of the Interaction Browser are:

4. The projection name is *Context*.
5. A list of contexts, one for each operation. *EllenAlarmCtx* is selected. A button toggles to the *Context Class browser*.
6. The Interaction diagram where the programmer works with Roles (move, select, link, add, remove, and rename) and their link structure. The FORECASTER Role is selected.
7. A list of FORECASTER RoleScripts, *checkWeather* is selected.
8. An editor for the selected RoleScript. (While a Squeak method accesses instance variables by name as declared in the class definition; a RoleScript accesses Roles by name as declared in the Interaction diagram.)

Figure 4: The Interaction Browser.
~~BB11PP-ContextInteraction-annotated.png~~