



A Walk Through the GBB Elevator Example

For x86 Computers Running Windows 95/98/NT

GBB Version 4.1

Overview

What this document is about

This document introduces you to GBB™ and its graphics capabilities by using the GBB elevator example. The GBB elevator example is a simple elevator-controller-and-simulator application that was written collectively by students in a GBB training course. During three days of hands-on training in using GBB, the students were able to design, develop, and deliver the application.

What the elevator example does

The elevator application is a basic blackboard application that acts as an intelligent dispatcher for a set of elevators in a building.

The application has two components: a simulator and a planner.

- **Simulator component**—Because there's no actual building containing the elevators, the simulator serves to simulate the behavior of the passengers using the elevators, as well as the elevators themselves. The simulator provides ongoing input to the planner component. Then, based on the tasks planned by the planner component, the simulator carries out the tasks by simulating the elevators' movements and actions.
- **The planner component**—The planner component plans the tasks that control the elevators' movements on the basis of the simulator's input. In response to the simulated actions of the passengers, the planner component plans the up and down movement of the elevators and the actions of the elevators, such as opening the doors.

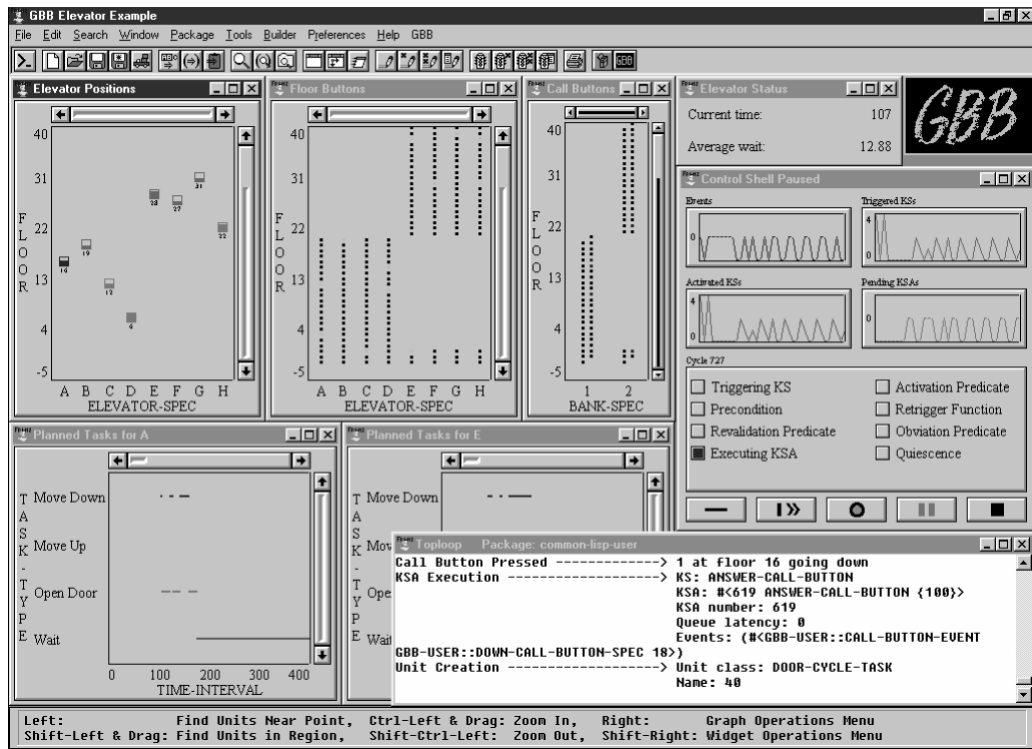
An "order" contains the information needed by the simulator to represent a person approaching a bank of elevator cabs, pressing an up or down button, waiting for the elevator doors to open, entering the elevator cab, pressing one or more destination-floor buttons, and then exiting the elevator cab at the desired floor. An initial set of elevator orders is generated by the simulator

component when the application starts up. Successive sets of orders are generated by the simulator in a random manner.

Time is represented by a clock that is an integer counter indicating the number of simulated seconds that have elapsed since the start of the application. The clock is incremented only after all simulation and planning activities (determining what to do and creating a task to handle it) have been completed for the current time tick.

Screen illustration

The following screen illustration shows the graphics windows displayed by the elevator application:



About GBB

To understand how the GBB elevator example performs processing and what the GBB Graphics System’s windows show, you should understand key concepts about GBB.

The blackboard database

A GBB application uses a global database called the **blackboard database**. This specialized, object-oriented database is a shared repository of problems, partial solutions, suggestions, and contributed information.

Blackboards and spaces

Within the blackboard database, **spaces** are the containers for storing objects. Spaces can be grouped into blackboards.

The elevator application uses three blackboards, in order to keep different kinds of data separate. One blackboard simply contains information about the building that houses the elevators, one contains information about the planning activities of the application, and one contains information about the simulation activities of the application. The blackboard containing simulation information, for example, has four spaces: one space stores elevator cabs, another stores simulated actions (that is, actions being performed by the elevator cabs), another stores orders, and yet another stores the current time.

Dimensions

A space is structured as a multidimensional volume. The space's **dimensions** provide a metric for positioning objects on the blackboard database in terms that are natural to the application domain. For example, in the elevator application, the space that stores objects representing simulated actions has five dimensions. These five dimensions are used for positioning objects representing:

- The time interval (duration) of the action
- The elevator cab that's performing the action (that is, the "simulated" elevator cab)
- The static elevator specification (used for planning the actions of the elevator cab)
- The floors the elevator will stop at or pass by
- An indicator of whether the simulated actions have been executed

Space dimensions can be **ordered**, which means they consist of a real-number interval, or **enumerated**, which means they consist of a set of labels. (Labels can be strings, symbols, or any other Common Lisp objects.) The position of an object along each space dimension is determined by the object's **dimension values**.

Dimensionality plays an important role in GBB. One of the most important uses of dimensions is for associative retrieval (for example, finding all the elevators in bank 1 that are going up or are waiting on a floor between floors 3 and 13). Dimensionality is also used to display blackboard information graphically.

Blackboard objects

In GBB blackboard objects are called **unit instances**. Each unit instance is a member of a **unit class**. The unit class defines information that is common to all instances of the unit class.

Unit instances are typically stored on spaces on the blackboard database—and typically positioned on the spaces based on their dimension values. (However, they can alternatively be stored outside of spaces on the blackboard database.)

Knowledge sources

Knowledge sources (KSs) are the software modules that provide specific expertise needed by a blackboard application. KSs interact in much the same way as human experts brought together to brainstorm a solution to a problem.

KSs can be implemented using widely differing approaches and programming languages. In the elevator application, all the KSs are programs written in Common Lisp; however, KS code can be any program or subroutine or even an expert system, a neural-net or fuzzy-logic routine, or an interface to a human operator.

The control shell

The **control shell** is the component of GBB that oversees the activities of the KSs. The control shell is like a human moderator overseeing a group of human experts that are cooperating to solve a problem; the control shell ensures that, at each step of the problem-solving process, the expert with the most pertinent knowledge will be the one to make a contribution. At each step, the control shell considers each KS's criteria for use, in order to apply the most appropriate KS to contribute to the solution.

The control shell triggers a KS in response to an **event** or events (for example, the creation of a unit instance) and then, if appropriate, activates the KS. Triggering the KS means making it a candidate for activation; and activation means creating a **KS activation** (KSA). The KSA represents the use of the KS in response to the particular event that triggered the KS. The control shell stores the KSA in a queue with other KSAs that are pending execution.

About the graphics windows

Before you start up the elevator example, it's helpful to be aware of the various types of graphics windows that will be displayed on your screen:

- **The GBB logo window**, in the upper, right-hand corner of your screen. The GBB logo window is an important mouse location, as the main graphics-system menus are available when the mouse is located over the logo window.
- **Five blackboard windows**, each displaying two-dimensional information about unit instances on the blackboard. These windows are illustrated and described in more detail below.
- **The Control Shell Window**, which enables you to monitor the operations of the control shell and interactively stop and restart it.
- **A status information window**, titled Elevator Status, which gives you information such as the current time (the number of simulated seconds that have elapsed since you started the application).
- **The Lisp listener window**, which displays a trace of all control-shell activities. Later in this tutorial, you'll examine the trace information displayed in this window.

These windows are illustrated and described in more detail on later pages.

Window arrangement

The arrangement of the windows depends on the resolution of your monitor and the size of your application window. If your monitor has sufficient resolution and a sufficiently large application window, all the windows described above will appear directly on your monitor.

On a low-resolution monitor or on a monitor on which the application window size is insufficient, a special Chain Manager window allows you to iteratively select among individual windows. For information about using the Chain Manager window, see “Considerations for small screen displays” at the end of this tutorial.

Running the elevator example

Initial activities

Start up the elevator example now.

Note: For step-by-step instructions on loading and starting the elevator example, see “Loading and running the elevator example” in the document titled *The Elevator Example*.

Notice that the Lisp listener window displays trace output indicating that system initialization is occurring. System initialization is the process that invokes the control shell. The application then creates the five blackboard windows, the Elevator Status window, and the Control Shell Window. It also adjusts the size of the Lisp listener window and displays mouse documentation in the mouse documentation window at the bottom of your screen.

Next, the application creates items such the elevator cabs (each positioned at the ground floor with its door closed), the elevator banks at which the elevator cabs are located, the call buttons (the up and down buttons at each floor), the floor buttons (the buttons inside the cabs), and the clock (the counter that keeps track of elapsed time). It creates these items by creating instances of the unit classes representing them.

Dynamic update

Notice that the elevator application dynamically updates each blackboard window. That is, as the elevator application runs, the blackboard windows change in response to the creation of unit instances representing the elevator cabs, floor buttons, and call buttons. Once these unit instances have been created, the blackboard windows will continue to change in response to changes to the unit instances’ attributes.

Planning and execution of elevator tasks

After all the unit instances have been created, the simulator component of the application begins simulating external events (such as the press of a call button and the movement of an elevator) and the planner component responds to those events by planning activities for each elevator and then sending commands to the simulator to execute each task.

Watch the call buttons and floor buttons light up and the resulting movement of the elevators. The simulator is placing orders for the elevators, the planner is planning the tasks that are causing the elevators to respond to the orders, and then the simulator is carrying out the tasks.

Exploring the graphics windows

Pausing the application

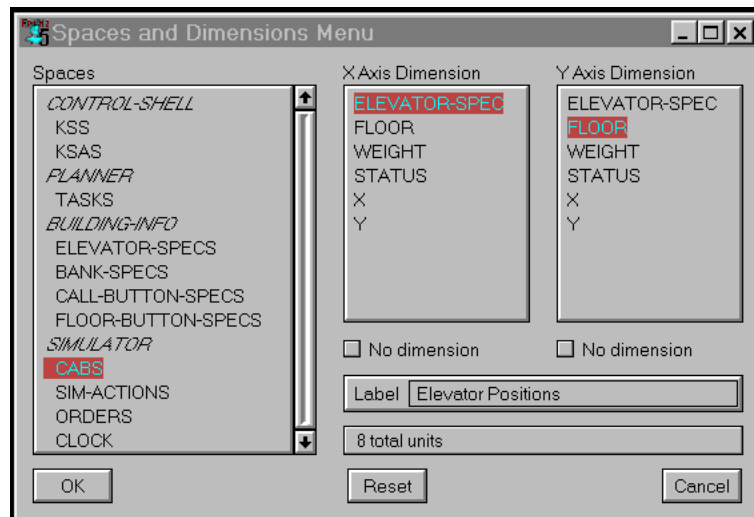
Let the application run until several elevators are in motion and then pause the application so you can look at the various graphics windows in detail. To pause the application, click on the pause/continue button in the Control Shell Window. The pause/continue button (||) is the second from the right, with the two vertical bars suggestive of the pause control on an audio tape or CD player. When paused, the title of the Control Shell window becomes “Control Shell Paused.” (To continue from the pause, you simply click on the pause/continue button again.)

About the blackboard windows

Blackboard windows are a special type of window provided by the GBB Graphics System to display a one- or two-dimensional view of one or more spaces or simply a listing of all unit instances stored on one or more spaces. Typically, this capability for presenting data in two dimensions is a very powerful feature in a GBB application.

Each of the five predefined blackboard windows in the elevator application shows a two-dimensional view of different information about the elevators. Using the space and dimension specifications for a window, the GBB Graphics System determines which unit instances are to be displayed in the graphics window. Because space dimensions, rather than specific unit instances, are associated with a window, GBB can dynamically update the unit instances shown in the window as unit instances are created, changed, or deleted.

The menu used for selecting a space and its dimensions for a blackboard window is shown below:

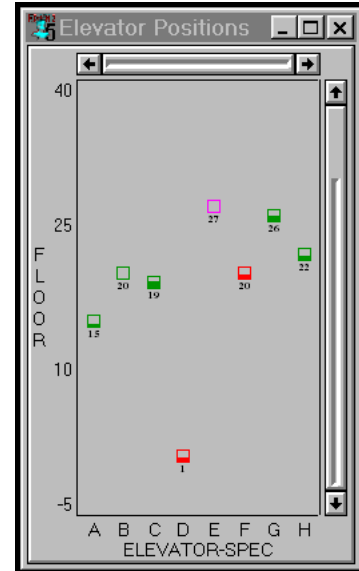


Elevator Positions Window

The Elevator Positions window displays all unit instances representing elevator cabs. The window shows the position of each elevator as it moves among the 43 floors in the building (the ground floor, two floors below ground, and 40 upper floors).

The icons represent the elevator cabs. The number below each icon indicates the floor at which the elevator is positioned. Inside each icon is a histogram indicating how full the elevator is.

Elevators moving up are green, and those moving down are red. Those waiting with an open door (for passengers to enter or leave) are magenta, and those waiting with a closed door are blue.



Note that the names for the x and y axes of the graph are the names of the space dimensions (`elevator-spec` and `floor`) being used to position the unit instances that are displayed.

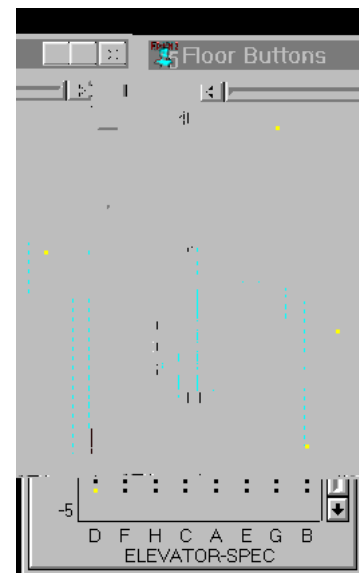
Looking at a unit instance's values

Later in this tutorial (see "Examining unit instances") you'll learn how to inspect the unit instances represented by these elevator cab icons. You'll find that the instances are of a unit class called `cab`; and they are stored on the `cabs` space, which has several dimensions in addition to the `elevator-spec` and `floor` dimensions shown in this window. (In fact, the dimensions called `weight` and `status` are used to draw the elevator cab icons in this window. The `weight` dimension determines how the histogram is drawn, and the `status` dimension determines the color of the icon.)

Floor Buttons Window

The Floor Buttons window displays all unit instances representing the floor buttons inside each elevator. The yellow dots indicate floor buttons that are currently lit.

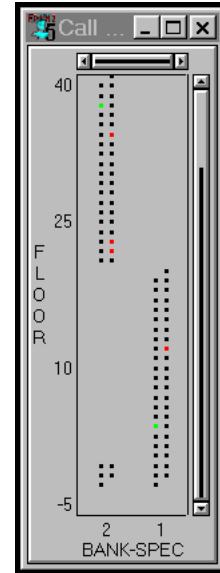
Notice that elevators in the first bank of elevators service floors 1–20 and those in the second bank of elevators service the two below-ground floors, the ground floor, and floors 21–40. Also notice that the unit instances representing the floor buttons are positioned on the `elevator-spec` and `floor` dimensions. These unit instances are stored on the space called `floor-button-specs`.



Call Buttons Window

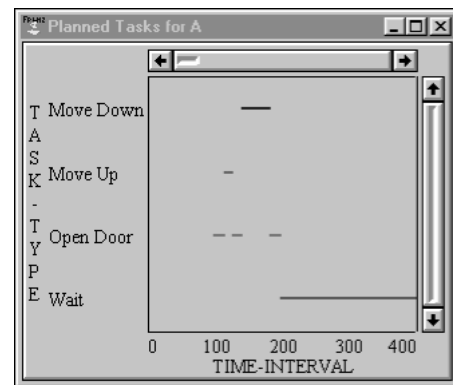
The Call Buttons window displays all unit instances representing the call buttons that are located at each bank of elevators, on each floor. The green dots (in the left-hand column) indicate “down” call buttons that are currently lit, and the red dots (in the right-hand column) indicate “up” call buttons that are currently lit. Notice that elevator bank 1 is made up of elevators A–D, and elevator bank 2 is made up of elevators E–H.

The unit instances representing the call buttons are positioned on the bank-spec and floor dimensions of the space called call-button-specs.



Planned Tasks Windows

The elevator example displays two Planned Tasks windows. The “Planned Tasks for A” window displays all unit instances representing planned tasks for elevator A. The planned tasks are shown by task type, displayed on a time line. Similarly, the “Planned Tasks for E” window shows information about the planned tasks for elevator E, displayed on a time line.



The unit instances shown in these windows are positioned relative to the time-interval and task-type dimensions, on the space called tasks. (Since the tasks take time to execute, the unit instances occupy an interval along the time-interval dimension; the bar displayed in the window indicates the start and end of the interval.)

The task types shown in these windows are as follows:

- **Move Down**—Shown in red, this task type indicates unit instances that cause the elevator to move downward.
- **Open Door**—Shown in magenta, this task type indicates unit instances that cause the elevator to open its doors, wait, and close its doors.
- **Move Up**—Shown in green, this task type indicates unit instances that cause the elevator to move upward.
- **Wait**—Shown in blue, this task type indicates unit instances that cause the elevator to wait with doors closed (since there are no other tasks currently planned for the elevator).

When the application starts up, the task that is planned for each elevator is to wait with doors closed; thus, this window shows a solid blue line from time 0 onward. As other tasks are planned, they are plotted on a time line; and, as each task is executed, it is deleted from the blackboard.

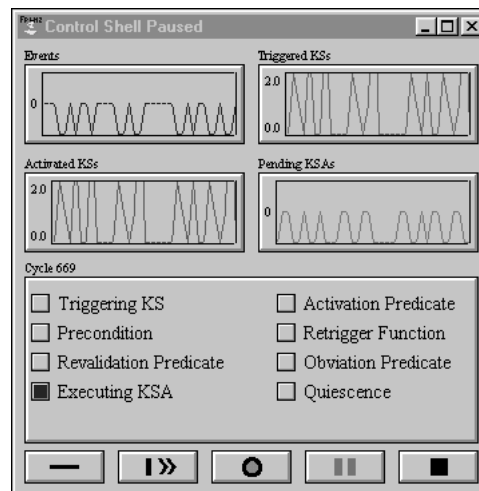
About the Control Shell Window

Monitoring information

The Control Shell Window provides the following information about control-shell operations:

- Historical information about events, triggered KSs, activated KSs, and pending KSAs. GBB updates these graphs during each cycle of the control-shell scheduling loop, to provide information about control-shell processing over time. (The scheduling loop is the activity during which the control shell processes events, determines which KSA is to be executed, and then executes the selected KSA.)
- The number of the current cycle of the control-shell scheduling loop.
- The current phase of control-shell processing; for example, triggering a KS or executing a KSA.

As shown below, the title of the Control Shell Window changes to reflect the current state of the control shell (in this case, paused). When a KSA is being executed, the name of the KS of the KSA is shown as the title of the window.



Note that, although the same colors are used in the graphs and in Blackboard Windows 4 and 5, there is no relationship between the historical graphs in the Control Shell Window and the task types shown on the time line in the Planned Tasks for A and Planned Tasks for E windows.

Control buttons

The buttons across the bottom of the Control Shell Window allow you to control the application's execution. The buttons allow you to:

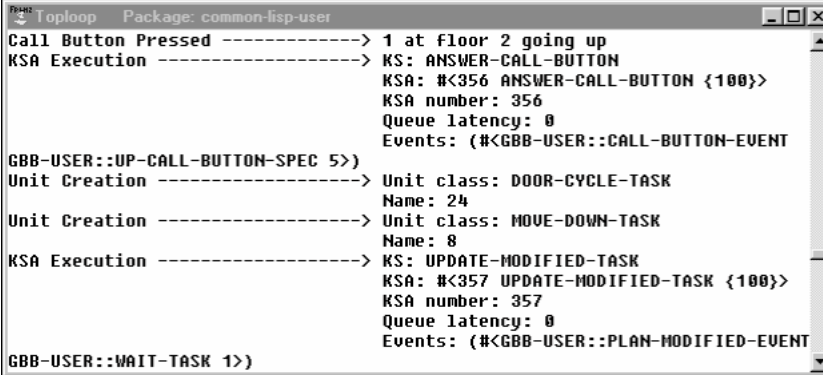
- Disable/reenable monitoring of the control shell (that is, stop or restart the phase indicator and graphs that indicate the control shell's activities)

- Choose stepping options and turn on stepping (thereby causing the control shell to pause at certain points during the scheduling loop)
- Suspend/resume printing of trace information about control-shell activities (in the Lisp listener window)
- Pause/continue the control shell
- Exit the control shell

About the Lisp listener window

Trace output

The elevator application displays a trace (or log) of control-shell operations in the Lisp listener window. The trace indicates events that have occurred and resulting control-shell actions.



```

Toploop Package: common-lisp-user
Call Button Pressed -----> 1 at floor 2 going up
KSA Execution -----> KS: ANSWER-CALL-BUTTON
                        KSA: #<356 ANSWER-CALL-BUTTON {100}>
                        KSA number: 356
                        Queue latency: 0
                        Events: (#<GBB-USER::CALL-BUTTON-EVENT
GBB-USER::UP-CALL-BUTTON-SPEC 5>)
Unit Creation -----> Unit class: DOOR-CYCLE-TASK
                        Name: 24
Unit Creation -----> Unit class: MOVE-DOWN-TASK
                        Name: 8
KSA Execution -----> KS: UPDATE-MODIFIED-TASK
                        KSA: #<357 UPDATE-MODIFIED-TASK {100}>
                        KSA number: 357
                        Queue latency: 0
                        Events: (#<GBB-USER::PLAN-MODIFIED-EVENT
GBB-USER::WAIT-TASK 1>)

```

Release the pause on the control shell now, to continue the application. Notice the detailed information displayed in the Lisp listener window as events occur and the control shell responds.

Performance improvement

The performance of the elevator application is limited by the time required to display the trace lines in the Lisp listener window. (In Allegro CL for Windows, the Lisp Listener window is an editable Windows window with a high character-insertion overhead.) If you want to improve the performance of the application, you can suspend the printing of trace output. (Later, you can resume printing of the output in order to examine the trace lines.)

How to suspend printing

To suspend printing of trace output, click on the center button at the bottom of the Control Shell Window (the button with the black circle icon, suggestive of the record button on an audio tape or CD player). Similarly, to resume trace-output printing, simply click on the button again.

Note: You can suspend printing of trace output at any time. If, after running the elevator application for a short time, you want better performance, you can suspend printing at that time. Similarly, you can resume the printing of trace output at any time.

Examining the blackboard windows

Valuable application information

During application development, GBB's graphics windows provide valuable visual feedback about your application, such as the progress of the application, and a quick means of diagnosing problems. For a delivered application, the GBB Graphics System provides the basis for a powerful monitoring and presentation user interface.

Wide range of features

The GBB Graphics System provides numerous monitoring facilities, display operations, and display options. As you move the mouse among the windows on your screen, the mouse documentation window provides short descriptions of the operations and options available. These standard GBB graphics features are available for any GBB-based application.

The sections below show you how to invoke some of the graphics operations that are available.

Note: If, while performing the activities in this tutorial, you get an error message indicating that there is not enough available memory to create a memory bitmap (a window), you need to increase your Windows virtual memory setting, quit one or more of your other Windows applications, or delete some other Lisp windows.

Pause the control shell now, so the application will be stopped while you invoke the graphics operations described below.

Examining unit instances

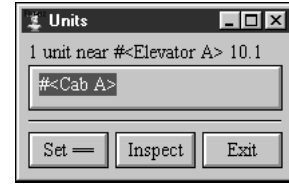
You can examine a specific unit instance shown in a blackboard window; or you can select a region of a window for which you want to examine unit instances, and then inspect the unit instances in that region individually.

Finding all unit instances near a point

To find the unit instance representing elevator cab A, click left on elevator cab A in the Elevator Positions window. Clicking left invokes the Find Units Near Point operation, which brings up a window that lists the unit instances (if any) near the point on which you clicked.

Remember: The mouse documentation window provides a handy source of context-sensitive descriptions of the functions of your mouse buttons or keystroke/mouse-button combinations.

Notice the heading in the window. For example, the heading 1 unit near #<Elevator A> 10.1 indicates that there is one unit instance near the point you clicked on (in this case, near the point on the elevator-spec dimension at which the Elevator A instance is located and the point on the floor dimension at which the value 10.1 is located). The unit instance shown in the window, Cab A, is the unit instance near the point you clicked on.



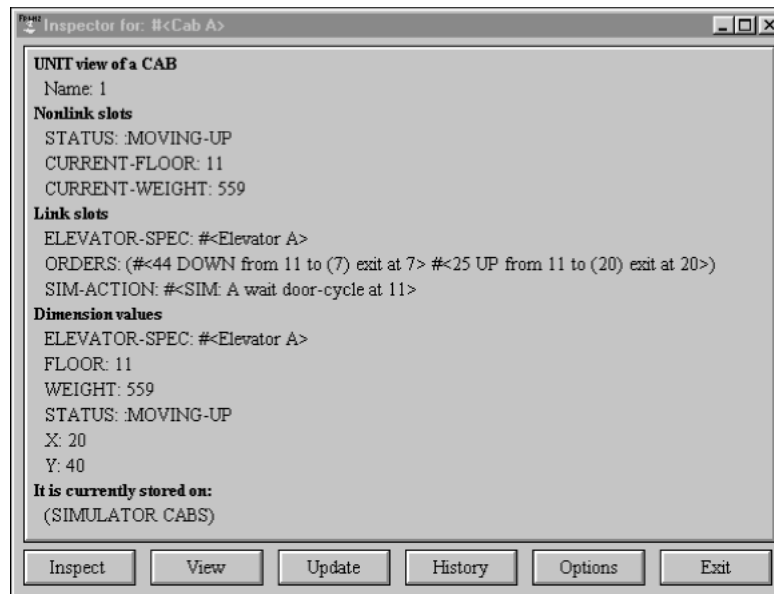
Note: Elevator A is an instance of the elevator-spec unit class, which is different than the cab unit class. The cab unit class represents the simulated elevator cabs themselves. To maintain a “clean” separation between the planner and simulator components, the planner component uses the elevator-spec unit class to plan future activities of the elevator cabs.

Inspecting a unit instance

Now that you’ve located the Cab A unit instance, you can inspect it to examine its values. By inspecting the unit instance, you’ll see the unit instance’s **link slots**, **nonlink slots**, and dimension values. Link slots contain relationships between unit instances. Nonlink slots contain other attributes of the unit instance. The instance’s dimension values determine its location on one or more spaces.

Inspect Cab A by clicking left on the Inspect button (since Cab A is already selected). GBB brings up an inspector window, which contains information about Cab A. Notice the nonlink slots, link slots, and dimension values of Cab A.

Tip: You may want to enlarge the inspector window so you can see all the values without having to scroll.



The inspector window shows you that Cab A has the following characteristics:

- Is an instance of the `cab` unit class.
- Is named 1. Unit instances have an implicitly defined slot, called `name`, that contains a unique identifier. No two instances of the same unit class have the same identifier. This identifier was generated by a default name-generation function, which simply increments the number for each additional instance that is created for the `cab` unit class. The name identifier is important because it allows GBB developers to retrieve unit instances by name, and to represent and restore unit links when saving and restoring blackboards.
- Has three nonlink slots, as follows:
 - **status**—Contains a keyword symbol indicating the whether the elevator cab currently has its doors open, is moving up, is stopped, or is out of service.
 - **current-floor**—Contains a number indicating the floor at which the cab is currently located.
 - **current-weight**—Contains a number indicating the current weight of the cab.
- Has three link slots, as follows:
 - **elevator-spec**—Contains the unit instance `Elevator A` (of the `elevator-spec` unit class), to which the `Cab A` unit instance is linked. (The `Elevator A` unit instance contains static information about cab A, such as its capacity and the floors it serves.)
 - **orders**—Contains a list of the orders that cab A is servicing.
 - **sim-action**—Contains the simulator action that cab A is performing.

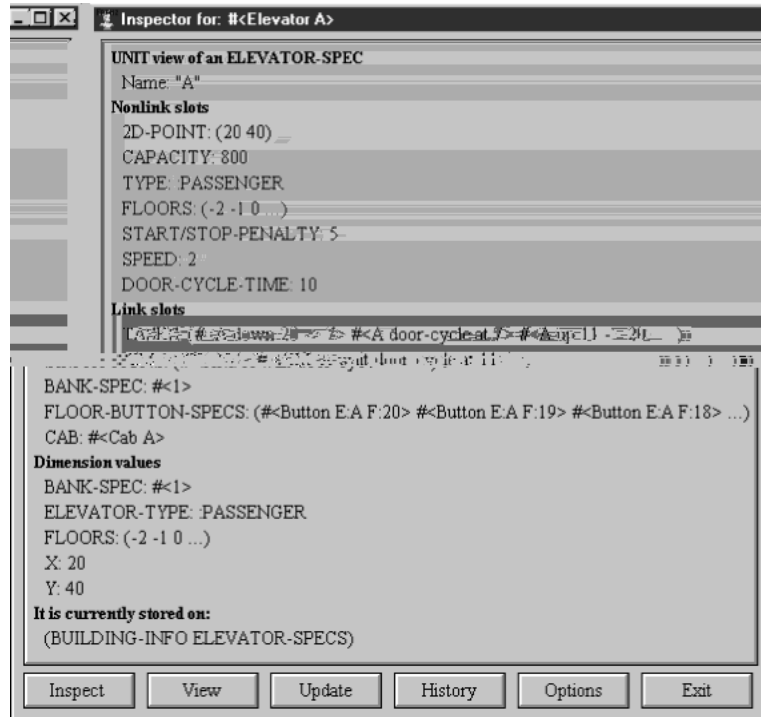
- Has several dimension values. These dimension values are obtained from the unit instance's slot values. For example, the value of the floor dimension is the value of the current-floor nonlink slot, and the value of the elevator-spec dimension is the value of the elevator-spec link slot. The values of the x and y dimensions are extracted from the unit instance stored in the elevator-spec slot.

Inspecting a slot

Now, inspect the elevator-spec link slot (containing the Elevator A unit instance). To inspect the slot, you could simply click left on it to select it and then click left on the Inspect button. However, use the following procedure instead:

- 1 Position the mouse cursor over the item ELEVATOR-SPEC #<Elevator A>, which is listed under the Link Slots heading.
- 2 Simultaneously press the Shift and Control keys and click left.

This procedure brings up a second inspector window, rather than overwriting the information in the first inspector window you created. Move the new inspector window off the other one and enlarge the new window.



The new inspector window displays information about the Elevator A unit instance. Note that the value of the cab link slot points back to the unit instance Cab A, to which the Elevator A unit instance is linked.

Also, note the nonlink slot called 2d-point, with the value (20 40). This slot is used to locate the values for the x and y dimensions of the Cab A unit

instance (as specified by a **dimension-value computation** that indicates where the values of the x and y dimensions for Cab A are to be found).

Quitting the inspection windows

Quit the unit instance and slot inspection windows by clicking left on the Exit button in the window. Then, quit the Units window by clicking left on its Exit button.

Finding all unit instances in a region

You can also retrieve all unit instances located in a region of a blackboard window (as opposed to retrieving unit instances near a single point). To mark a region, you simultaneously press the Shift key and click left, drag the mouse to draw the region, and release the button.

Now, in the Planned Tasks for E window, mark a region that contains part of at least one of the bars. GBB will bring up a window displaying a list of all unit instances positioned in the region you specified.



From the Units window, you can inspect the unit instances individually. Notice that the sequence of tasks that have been planned is evident by looking at the value of the `previous-task` and `next-task` link slots and then inspecting those slots to see the value of the linked slots. Also notice that the `task-type` dimension is an enumerated dimension for which the value is a symbol (for example, “Open Door” or “Move Up”).

Zooming in on unit instances

The GBB Graphics System enables you to zoom in on any area of a blackboard window. Zooming in enables you to see in more detail the unit instances positioned in that area of the space associated with the blackboard window (that is, relative to the extent of the dimensions represented in that part of the window).

Also, zooming in sometimes helps you read the dimension labels displayed in the window (for example, if they are truncated in the window).

Zoom In operation

Try zooming in on planned tasks displayed in the graph in the Planned Tasks for A or Planned Tasks for E window. By zooming in, you see a more focused view of the task unit instances in the area you select.

Perform the following steps to zoom in on a portion of the graph displayed in the blackboard window:

- 1 Position the mouse cursor at the top, left-hand corner of the rectangular area of the graph you want to zoom in on.
- 2 Simultaneously press the Control key and click the left mouse button to invoke the Zoom In operation.
- 3 Holding the mouse button down, drag the mouse to the lower, right-hand corner of the area you want to zoom in on, and release the mouse button.

You can now more clearly determine the time interval for which tasks are planned.

Zoom Out operation To zoom out, simultaneously press and hold down the Shift and Control keys and click right.

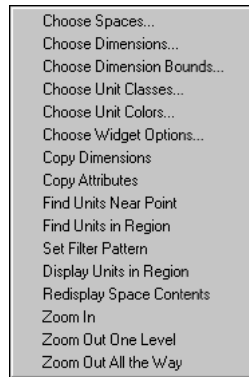
Note: If you zoom out again, the entire extent of the time-interval dimension will be displayed, since you zoomed all the way out. (By default, the window is set to display the time interval 1–400.) You can zoom back in to the same area, if desired.

Zoom to Fit operation Try using the Zoom to Fit operation, which you initiate from the Graph Operations Menu (described below). Zoom to Fit finds the bounds of the data that is to be displayed or is already being displayed and displays or redisplay the data with its bounds. For example, if you've already zoomed in on part of a dimension (by using Zoom), Zoom to Fit finds the whole piece of that dimension, based on its bounds, and redisplay the whole piece.

Using pop-up menus

So far, you've used mouse clicks or mouse click/keystroke combinations to perform various graphics operations. Those operations, as well as many others, are accessible from pop-up menus provided by the GBB Graphics System. Each pop-up menu displays a related set of graphics operations.

Graph Operations Menu One type of pop-up menu is the Graph Operations Menu. You can use the Graph Operations Menu to invoke operations that are specific to blackboard windows (such as Find Units operations and Zoom operations).



To bring up the Graph Operations Menu, click right on a blackboard window. Notice the many options you have for manipulating the blackboard window and examining the data that is displayed. If you want to try any of these operations, you can use the mouse documentation to guide you.

Other menus

There are several other types of pop-up menus. Some are accessible from the blackboard windows and the Control Shell Window, and others are accessible only from the GBB logo window. To explore these other types of pop-up menus, refer to the mouse documentation for help.

Examining KS and control-shell activities

Trace information

Release the pause on the control shell now, to continue the application. Notice the trace information in the Lisp listener window. As GBB runs the KSs, it displays descriptive lines in the Lisp listener window about the events signalled by the KSs.

Note: If earlier in this tutorial you suspended the printing of trace output in order to improve the performance of the elevator example, there won't be many lines of trace output in the Lisp listener window. Resume printing now by clicking on the center button at the bottom of the Control Shell Window (with the black circle icon).

It's difficult to read the trace information as it's being displayed. However, you can cause the control shell to pause at specified intervals, to allow you to read the trace information. This process, described below, is called **stepping**. Stepping is actually intended for use in debugging a GBB application.

Pausing processing at specific intervals

The control shell allows you to "step" through processing, which means that the control shell pauses at specified points in the scheduling loop (for example, before a KS is activated) and awaits your interaction. Although

stepping is intended as a debugging aid, you can use it now to enable you to more easily read the trace lines displayed in the Lisp listener window.

Stepping through control-shell processing

Turning on stepping

You turn on stepping from the Control Shell Window. Perform the following steps:

- 1 Click left on the step options ($|>>$) button. GBB displays a menu of control-shell stepping options (after it finishes drawing all the windows).
- 2 Turn on stepping for KSA execution by clicking left on **KSA execution** and on the OK button.

The control shell pauses before executing the next KSA, and a dialog titled “Control Shell Stepper” pops up (positioned where your mouse cursor is).



Note: If the application is paused, you must resume it before the Control Shell Stepper dialog will appear.

Move the Control Shell Stepper dialog to another location on your screen, to uncover the Lisp listener window. Then, enlarge the Lisp listener window so you can see many trace lines at one time.

When you're ready for the control shell to continue, click left on the Continue button in the Control Shell Stepper dialog. To continue subsequent stepping through control-shell operations, click on the Continue button after each pause.

Turning off stepping

To turn off stepping, you simply click left on the Quit button in the Control Shell Stepper dialog. The control shell will no longer pause during processing.

Leaving the elevator example

To quit the elevator application:

- 1 Bring up the GBB Graphics System menu by clicking left on the GBB logo window in the upper right corner of your screen.
- 2 Click on **Exit Elevator Example**.

This procedure exits the control shell and closes the blackboard windows, Elevator Status window, and Control Shell Window. If you are running the stand-alone elevator demo, this procedure also exits Lisp.

Other considerations

The sections below explain the following conditions:

- Graphics windows that are overlaid
- Error recovery in Allegro Common Lisp

Considerations for small screen displays

If you run the elevator example on a screen with resolution of 800 x 600 pixels or less or if the size of your application window is insufficient to display all the graphics windows, the configuration of the windows that are displayed is different than on a larger screen.

Chain Manager window

On a small display, the GBB graphics windows remain overlaid, and you must use the Chain Manager window to access the underlying windows. (The Lisp listener window is always displayed on your screen, however.) The Chain Manager window contains several buttons, as described in the table that follows:

Click left on this button	To perform this action
Previous	Bring to the front the preceding graphics window. (If the Floor Buttons window is displayed, the previous window is the Elevator Positions window.)
Next	Bring to the front the immediately following graphics window. (If the Planned Tasks for E window is displayed, the next window is the Elevator Positions window.)
Select	Bring up the Chained Windows menu and select the one that is to be brought to the front.
Remove	Bring up the Chained Windows menu and select the window or windows you want to remove from the chain. Removed windows are displayed on the screen along with the chained window that is at the front of the chain.
Add	Bring up the Chained Candidates menu and select the window or windows you previously removed from the chain and want to add back into the chain.

Error recovery

When using Allegro Common Lisp

If you're running the elevator application on Allegro Common Lisp and either of the following conditions apply, the problem described below can occur:

- You're running Lisp on UNIX and not using the emacs interface
- You're running Lisp on Windows and not using either the emacs interface or IDE in Allegro CL

When an error is signalled, the error process and the normal Lisp listener process are both trying to read from the same stream. (More specifically, the debugger and the elevator example are trying to read input at the same time, and they succeed in reading only alternate lines.)

To exit from an error, enter `:pop` twice in the Lisp listener window to ensure you're really out of the error.

For more information

About the elevator example

For information on the elevator application, see the document titled *The Elevator Example*. This document describes the blackboards, unit classes, and spaces used by the application; provides static data about the elevators and elevator banks; and lists the initial set of elevator orders generated by the simulator component.

About the GBB Graphics System

To learn how to create graphical blackboard windows in a GBB application and customize them to display the information you need, see the tutorial titled *How to Use the GBB Graphics System with the Limo Example*. For information on how to use your mouse with the GBB Graphics System and how to use the pop-up menus and dialogs, see the *Interacting with ChalkBox* manual.

About GBB

For more information about the GBB concepts discussed in this tutorial, see the GBB documentation set.

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Knowledge Technologies International, Inc.

U.S. Government Restricted Rights Legend: Use, duplication, and disclosure by the Government are subject to restrictions of Restricted Rights for Commercial Software developed at private expense as specified in DOD FAR 52.227-7013 (c)(1)(ii).

Copyright © 2000 by Knowledge Technologies International, Inc. All rights reserved.

GBB, ChalkBox, KTI Tools, and the KTI logo are trademarks of Knowledge Technologies International, Inc. Other brand or product names are trademarks or registered trademarks of their respective holders.

May 2000

Knowledge Technologies International, Inc.
401 Main Street
Amherst, MA 01002
Phone: (800) 577-8990, (413) 256-8990
E-mail: gbb-info@KTIworld.com
Fax: (413) 256-3179
WWW: www.KTIworld.com



KNOWLEDGE
TECHNOLOGIES
INTERNATIONAL

www.KTIworld.com