# The Playground Mediator: A Visual Tool for Configuring and Debugging Distributed Applications

T. Paul McCartney

WUCS-98-09

March 1998

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

# Mediator

## User Manual

T. Paul McCartney

---

*Revised for **Mediator v1.3.2***
*March 1998*



*Distributed Programming Environments Group*
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

---

### Abstract

The Mediator is a visual configuration tool for use with The Programmers' Playground distributed programming environment. With the Mediator, one can interactively launch distributed application modules, configure communication among the modules, observe communication patterns, interactively control module communication, kill running modules, and receive imported applications from a separate World Wide Web interface. This manual describes how to use the Mediator both as a stand-alone configuration tool and as a visual interface to the Playground Application Management System.

# Table of Contents

# 1 Introduction

The Programmers' Playground is a software library and run-time system for creating distributed multimedia applications [1], [2]. A distributed application consists of a set of communicating *modules* (i.e., processes), written as C++ programs, using special data types from the Playground library. Variables declared using these data types can be *published*, making information available to external modules. The communication structure among the modules of a Playground application is defined by a set of *logical connections* among published variables of the modules. Communication among modules of a distributed system occurs implicitly; when a published variable is modified, the new value of the variable is automatically sent to its connected variables. In this way, each module can be created independently of the modules with which it communicates.

See the Playground Veneer Reference manual [2] for more information on the Playground component model.

This manual describes how to use a visual configuration tool called the *Mediator*. The Mediator is primarily an application design tool that enables Playground distributed applications to be constructed using a visual interface. With the Mediator, one can interactively launch the modules of a Playground distributed application, configure the communication among the modules dynamically at runtime, and kill running modules.

The remainder of this manual is organized as follows. Section 2 describes the Mediator's basic operations such as establishing a communication configuration. This section focuses mainly on the Mediator as a stand-alone visual configuration tool. Section 3 discusses how to use the Mediator in conjunction with the Playground Application Management System. Application Management enables interactive launching of applications, launching and importing applications through a World Wide Web interface, and making completed applications available to others on the World Wide Web. Section 4 discusses probes, a distributed debugging mechanism. Section 5 lists and describes the Mediator's supported command line arguments.

# 2 Visual Configuration

The Mediator provides a graphical user interface for viewing and configuring the modules of a Playground distributed application. Figure 1 shows the Mediator with two active modules and a single bidirectional connection.

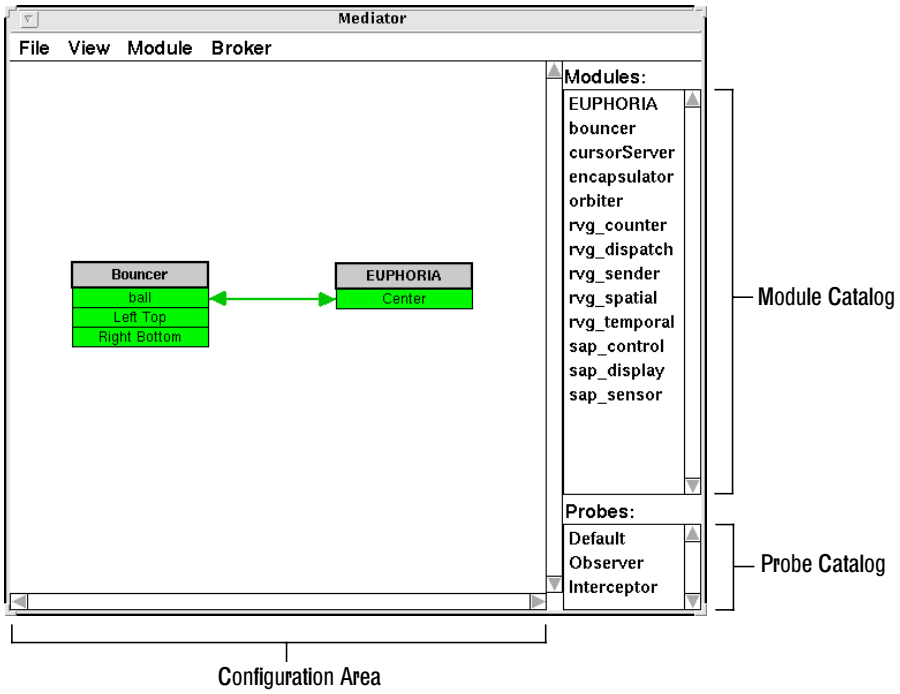


Figure 1: Mediator window.

The Mediator is started with the following shell command. See Section 5 for a list of supported command line arguments.

```
PGmediator    (on Windows: start PGmediator)
```

## 2.1 Modules

In the Mediator, active Playground modules are represented visually as boxes within the *Configuration Area* (see Figure 1). This visual representation includes a "title bar" containing the module name and a list of published variables. For example, Figure 2 shows a module named "Bouncer" with three published variables.

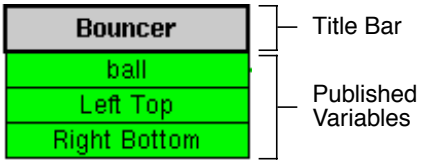


Figure 2: Module.

Variables are colored according to their data type. Table 1 lists the colors associated with each Playground data type.

Table 1: Data type colors.

| Data Type | Color |
|---|---|
| PGint | Magenta |
| PGreal | Cornflower Blue |
| PGbool | Yellow |
| PGstring | Salmon |
| PGmemoryBlock | Tan |
| PGtuple | Green |
| PGarray | Orange |
| PGmapping | Cyan |
| PGlist | Sky Blue |

### *How modules appear in the Mediator*

Obviously, you would not want to see every running Playground module in the world. Instead, the Mediator views only certain modules on which you are currently working. A running module can be incorporated into the Mediator's visual interface in a number of ways:

- The module is started from the shell, using the Mediator's `.pginitrc` file [2].
- The module is started from the Mediator, interactively from the Module Catalog.
- The module is started from the Mediator, by loading a saved configuration (see Section 3.3).
- The module is already running, and is imported into the Mediator using the Application Management System (see Section 3.2).

As described in the Playground Veneer manual, a *parent module* is a module that receives information about *child modules*. The Mediator is a parent module, which visualizes information about its child modules. When the Mediator is started, it writes its communication ID to a special file called "`.pginitrc`." When another module is started in the shell (i.e., by simply executing the module's binary), the module reads the ID and uses it to connect to the Mediator.

Starting modules with the Application Management System frees users from the task of starting each module from the shell. All but the first option above require the use of the Application Management System. Assuming that you have Broker and Launcher modules running (see Section 3.1), the Mediator's *Module Catalog* (Figure 1) contains a list of modules that you can start through the Application Management System. A module can be launched by dragging its name from the Module Catalog to the Configuration Area.

### *Selecting and moving modules*

A module can be selected by clicking on its title bar. Selected modules are shown in inverse colors. Clicking in the background of the configuration area deselects all currently selected modules.

A module can be moved by dragging its title bar.

### *Killing modules*

Selected modules can be killed by pressing the **backspace** or **delete** key or by choosing the **Kill Selected** item from the **Module** menu. This removes the modules from the Mediator and sends a Playground kill request to each module. Note that the implementor of a Playground module can optionally have the module handle a kill request, refusing to actually terminate.

### *Removing modules*

Selected modules can be removed from the Mediator, without being killed, by choosing **Remove from View** from the **Module** menu.

## 2.2 Connections

The Mediator visualizes logical connections among visualized modules. Both connections initiated by the user and by an external source (e.g., through a Veneer API connection request) are shown. Connections to implicit Playground variables (e.g., a module's "presentation variable") and to modules that do not appear in the Mediator are not shown.

### *Making connections*

Connections are formed among modules by dragging from "source" published variables to "destination" published variables. The following buttons are used to control connection properties:

- **Left mouse button** - form a unidirectional connection.
- **Middle mouse button** - form a bidirectional connection.
- **Shift key** - connection does NOT have the "send on connect" property [2].

By default, all connections are made with the "send on connect" property unless the Shift key is held down.

As the user specifies a connection, the Mediator performs permission and type checking to determine if the variables are compatible. When the user selects a variable as a source, the Mediator checks if the variable is a valid source (i.e., it must have "read" permission). If so, the variable is highlighted and a proposed connection arrow appears, as in Figure 3.

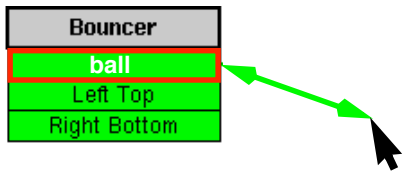


Figure 3: Dragging a connection with the mouse.

As the connection arrow is dragged over variables of other modules, each variable is highlighted if and only if it is a valid destination of the connection (i.e., the types and permissions of the source and destination are compatible). Releasing the mouse over a compatible destination variable causes a connection request to be sent to the endpoint modules. Pending connections are shown as dashed arrows until the connection has been approved by each module; successful connections are shown as solid arrows (see Figure 1).

### *Deleting connections*

A connection can be deleted by clicking on it with the **right mouse button**. Doing this causes the connection to be colored red and a disconnect request to be sent to the endpoint modules. Later, the connection is actually removed.

## 2.3 Configuration View

The Mediator's view of modules and connections can be zoomed in/out and panned.

- Choosing **Zoom In** or **Zoom Out** from the **View** menu increases/decreases magnification by 10%.
- Shortcut to zoom in/out menu items: pressing "**=**" or "**-**" on the keyboard.
- Choosing **Super Zoom In** or **Super Zoom Out** from the **View** menu increases/ decreases magnification by 50%.
- Shortcut to super zoom in/out menu items: pressing "**+**" or "_" on the keyboard.

The visible configuration area can be panned using its right and bottom scroll bars.

# 3 Using the Application Management System

The Application Management System [5] is a separate tool provided with Playground that is used to launch the modules of a distributed application and to (optionally) make the application available to others on the World Wide Web. This chapter describes the basics of using the Application Management System as it pertains to the Mediator. More detailed information can be found in the Application Management User Guide [5].

## 3.1 Using the Broker and Launcher(s)

The Application Management System's *Broker* and *Launcher* are used in launching Playground modules. The job of the Launcher is to launch modules on a particular computer or to delegate module launching to a "sublauncher." The role of the Broker is to decide which launchers to use in launching an application, to configure communication among launched modules (i.e., when NOT using the Mediator), and to communicate with other system components such as the *Liaison* [5], *Application Daemon* [5], and the *Mediator*.

In order to use the Application Management System, you must start at least one Launcher locally and a Broker. See the Application Management User Guide [5] for instructions and options for starting these components.

## 3.2 Mediator-Broker Communication

The Mediator connects to the Broker in order to receive information about available modules, to send module launch requests, and to receive active modules. If the communication between the Mediator and the Broker is successful, entries should appear in the Mediator's "Module Catalog" table (see Figure 1). These entries represent available modules that can be launched by the Application Management System. To add your modules to this list, see the portion of the Application Management System manual describing how to add modules to a "launch tree."

### *Launching a module*

A module launch can be specified in the Mediator by dragging a module entry from the Module Catalog and placing it within the Configuration Area. This creates a placeholder for the module, as shown in Figure 4.


Figure 4: Module placeholder, in "Launching" and "Connecting" stages.

If successful, a module placeholder goes through two stages: "Launching," when it is waiting for the Broker to launch the module, and "Connecting," when it is waiting for information about the module's variables and connections. After this process is complete, the placeholder is replaced by a visualization of the module.

If the placeholder disappears before entering the "Connecting" stage, the Application Management System was unable to launch the module and has returned an error. If the placeholder remains for

an extended period in the "Launching" stage, there is probably a problem with the Application Management System itself.

### *Customizing module launch properties*

A number of module launch properties can be specified both before and after a module is launched. Holding down the **control key** when launching a module brings up a dialog box for specifying these properties, as shown in Figure 5. Holding down the control key and clicking on the title bar of an existing module that has been launched through the Mediator will also expose this dialog box.

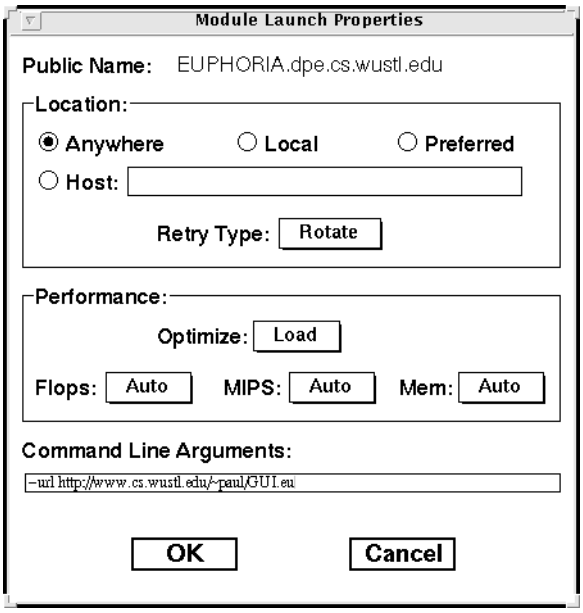


Figure 5: Module launch properties dialog.

The "location" section is used to specify a preference to the Broker concerning where to launch the module. The "retry type" specifies how the Broker should retry a module launch, in the event of a launch failure.

The "performance" section is used to specify preferences concerning the module's performance requirements. This information is used to decide onto which computer the module should be launched. For example, one may specify that a module should be placed on a computer with good FLOPS (floating point operations) performance with a minimal load.

The "Command Line Arguments" section is used to specify command line arguments for modules at launch time. For example, in EUPHORIA [3], specifying the "-url" or "-file" options are helpful in creating an application specification that can be launched through the World Wide Web (see Section 3.3).

See the Application Management System manual for more information on these options.

### *Connecting & disconnecting from the Broker*

Under most circumstances, the Mediator and the Broker connect automatically. When the Mediator starts, it checks your `~/.pgdir/Broker.Active` file to get the communication ID of your Broker, if one exists. You can manually disconnect and reconnect with the Broker by choosing **Disconnect from Broker** and **Connect to Broker** from the **Broker** menu. Disconnecting also has the effect of deleting all pending module placeholders from the Mediator view.

### *Launching and importing applications from the Web*

As described in the Application Management System manual, complete Playground applications can be launched or imported through the World Wide Web using an applet called the *Liaison*. These modules are automatically viewed in your Mediator, provided that your Mediator is connected to the same Broker that is being used by the Liaison.

## 3.3 Saving & Restoring Applications

It is possible to save a configured application by choosing **Save...** from the **File** menu. Choosing **Load...** from the **File** menu allows you to later launch modules, and restore the module positions, pan/zoom, and the connections among the modules.

Upon loading, modules may be launched, depending on how they were started in the saved configuration. If a module was started externally from the Mediator (e.g., from the shell), the Mediator assumes that it will be started externally again before or after file loading. If a module was launched from the Mediator, the Mediator will launch the module upon loading the configuration.

Externally launched modules are identified by their module name when they connect to the Mediator. If multiple modules have the same name, there can potentially be problems. Starting modules through the Mediator is recommended.

In order to load a configuration in which modules will be automatically launched, the Mediator should be connected to a Broker that has the modules available.

### *Clearing the load monitor*

The “load monitor” waits for modules and published variables that were specified in a loaded file to appear in the Mediator. Once these appear, the Mediator is able to request connections to be made among the modules. If there is a problem or if you wish to terminate a load that is in progress, choose **Clear Load Monitor** from the **File** menu.

### *Saving an Application Specification*

The *Application Daemon* is a component of the Application Management System that is used to make Playground applications available on the World Wide Web. In order to do this, an “application specification” is required. This can be generated by saving a configured application from the Mediator using the filename `new.spec`. Each module in the application should have been started through the Mediator (i.e., launching from the shell is not allowed).

Additionally, a *join.spec* file is also used by the Application Daemon for client/server applications. See the Application Management System manual for how to generate this file.

# 4 Probes

To facilitate end-user debugging of Playground distributed applications, the Mediator provides a mechanism known as a *probe*. A probe is an object that can be placed on a connection for the purposes of monitoring and controlling its associated communication. With a probe, end-users can observe when values are sent between modules and can slow down or control the rate that values are delivered.

Two types of probes are currently supported by the Mediator: *observer probes* and *interceptor probes*. An observer probe is passive and does not interfere with a connection's communication. An interceptor probe, as its name suggests, changes the communication structure between modules so that it intercepts values that are sent. While this results in some degradation in performance, interceptor probes give users the ability to interactively control the delivery of values between modules. An observer or interceptor probe is associated with a connection by dragging the appropriate probe type from the probe catalog (Figure 1) onto a connection. A probe is shown graphically at the midpoint of the connection as an "LED," as shown in Figure 6.

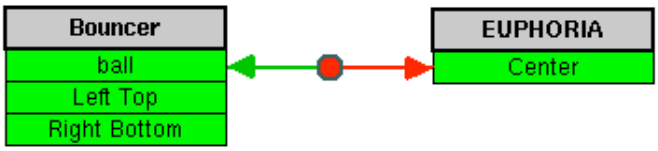


Figure 6: A probe monitoring communication between modules.

## *Animation*

When a value is sent across a connection and received at the probe, the probe blinks red (downstream communication) or green (upstream communication). Note that if values are constantly being received, the probe will remain solid red or green. For interceptor probes, received values are intercepted by the probe and are later forwarded out to the receiver module. Forwarding of a value from a probe is shown by coloring the segment of the connection from the probe to the receiver with the color red.

## *Flow Control*

By default, an interceptor probe forwards values to the receiver module immediately. However, the user may choose to block delivery or to change the delivery rate for debugging purposes. When this occurs, the interceptor queues received values for later delivery. From the probe's pop-up menu, choosing **Suspend** blocks the forwarding of values. Graphically, a suspended probe is shown with an "X" over it. Choosing **Set Delay** sets the rate at which queued items are delivered.

Interceptor probes only queue a maximum of 100 values before ignoring delays or blockage.

In either of the suspended or blocked states, value delivery can be controlled interactively using the left mouse button (see below). Values may be delivered one at a time or the entire queue of values can be flushed, delivering all values.

The following buttons are used to control a probe:

- **Left mouse button** - deliver the next value received (interceptor only).
- **Left mouse button, double click** - deliver all values received (i.e., flush, interceptor only).
- **Middle mouse button** - activates the pop-up menu for selecting various options.
- **Right mouse button** - delete the probe.

## 4.1 Probe Dumper Module Interface

The Mediator allows probe updates to be exported to an external module known as a *probe dumper*. This feature is useful for creating external modules for debugging or visualizing the data traffic of an application. A module must implement the "probe dumper interface" (i.e., published variables and behavior) that the Mediator expects in order to be used as a probe dumper (see below).

A module is designated as a probe dumper in the Mediator by selecting it and choosing **Set Module as Dumper** from the **Connection** menu. All probes created afterwards will send copies of their updates to the designated probe dumper module.

A sample probe dumper module called "PGprobeDumper" is included with the Playground distribution. This module simply prints the values sent across each probe to standard output. The source code for this module is provided in the "src/util" directory of the Playground Developer's Edition distribution. This code can be easily modified for other purposes.

The following is a summary of how to create a custom probe dumper using the supplied example. See the Playground Veneer Manual [2] for more information on how to implement modules.

- Create a module including the PGpoint, ProbeDumperNew, and ProbeDumperTime classes.
- Publish a write-only variable called "New Variable" using the ProbeDumperNew class.
- Modify or create a subclass of the ProbeDumperVariable class that has the same published interface. This class will be used to receive the values that a single probe observes along its connection. The class should implement the desired behavior for reacting to these updates.
- Upon receipt of a new value in "New Variable," create a variable using the ProbeDumperVariable class (or class that implements the same interface). Set a reactor for this object (i.e., set itself as its own reactor).
- Call the "Connect" method of "New Variable" to connect from the sender module variable to the newly created ProbeDumperVariable.
- React to each change to each created ProbeDumperVariable.

# 5 Command Line Arguments

Optional command line arguments allow users to customize the execution of the Mediator.

Table 2: Optional command line arguments.

| argument | default | description |
|---|---|---|
| **-buffer** | 580x400 | Size of offscreen buffer, used for screen updates. |
| **-colorDelta** | 200 | Maximum color approximation distance in RGB space. |
| **-display** | no default | X windows display name [4]. |
| **-file** | no default | Saved configuration to load upon startup. |
| **-invalidAreas** | 3 | Number of invalid rectangles maintained. |
| **-pollDuration** | 50 | Event polling time before drawing, in msecs. |
| **-pollSleep** | 10 | Sleep time while polling for events, in msecs. |
| **-showSpecial** | false | View "special" modules such as PGbroker, PGlauncher, and PGappDaemon in the Mediator. |

For example, to start the Mediator with specific display and a small buffer:

```
PGmediator -display kite.cs.wustl.edu:0.0 -buffer 200x200
```

## Double Buffering

Double buffering is used for smooth, flicker free, graphics rendering. This means that a resource called a "pixmap" must be allocated to buffer intermediate drawing results. The size of the pixmap is determined by the `-buffer` argument. Setting this value to a large size can result in more efficient drawing. Unfortunately, large pixmaps use a lot of memory; setting this value too large can cause the Mediator not to start due to lack of memory, giving an X-windows error.

## Color Allocation

Workstations that have a limited number of colors (e.g., 8 bit depth or 256 simultaneous colors) can have problems managing how colors are allocated. The Mediator controls how color is allocated, and can approximate a requested color to an already allocated color. Color approximation degree is set by the `-colorDelta` option. Color delta is the maximum distance in RGB space in which two colors can be considered equivalent. Setting this value lower will tend to match the requested values more exactly (e.g., setting color delta to 0 disables color approximation).

## Invalidation

Multiple "invalid areas" can be maintained for the Mediator window. These areas determine which portions of the window need to be redrawn when the appearance of window items change. Having more invalid areas is likely to make drawing more efficient if the buffer is small or many sparsely positioned, disconnected graphics items change sporadically. However, on a workstation with fast graphics capabilities, fewer invalid areas may result in more efficient drawing.

### Event Loop

The Mediator's event loop is timed according to the `-pollSleep` and `-pollDuration` arguments. Before drawing is performed in an iteration of the event loop, the system first polls for events and updates from the Playground environment. The polling time is determined by poll duration. This allows the system to gather many changes to draw simultaneously, rather than drawing each change separately. The duration effectively determines the maximum "frames per second" update rate of the drawing. The default setting allows for at most 20 updates per second; setting this value higher can result in more efficient, but "jumpier", drawing. During the polling loop, the Mediator repeatedly sleeps for a period of time (determined by loop delay) to wait for new events and to give other processes a chance to run. Setting this value lower can result in faster drawing. However, this can cause the Mediator to monopolize the workstation's CPU.

# 6 Acknowledgments

This research was supported in part by National Science Foundation grants CCR-91-10029, CCR-94-12711, and ARPA contract DABT63-95-C-0083.

# References

[1] Kenneth J. Goldman, Bala Swaminathan, T. Paul McCartney, Michael D. Anderson, and Ram Sethuraman. The Programmers' Playground: I/O Abstraction for User-Configurable Distributed Applications. IEEE Transactions on Software Engineering, 21(9):735-746, September 1995.

[2] Kenneth J. Goldman, Joe Hoffert, T. Paul McCartney, Jerome Plun, Todd Rodgers. Building Interactive Distributed Applications in C++ with The Programmers' Playground. Washington University Department of Computer Science technical report WUCS-97-14.

[3] T. Paul McCartney, Kenneth J. Goldman. EUPHORIA Reference Manual. Washington University Department of Computer Science WUCS-97-13, February 1997.

[4] Robert W. Scheifler, James Gettys. X Window System, Third Edition. Digital Press, 1992.

[5] William M. Shapiro, T. Paul McCartney, E.F. Berkley Shands. The Programmers' Playground Application Management System User Guide. Washington University Department of Computer Science WUCS-97-32, August 1997.