



# ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST



California State University, Sacramento's

**PC<sup>2</sup>**  
Version 8.5

# Contest Administrator's Installation and Configuration Guide

<Last Update: 2003-07-29>

# Table of Contents

<b>1. Overview</b> .....	<b>1</b>
<b>2. PC<sup>2</sup> Startup Checklist for Geniuses</b> .....	<b>2</b>
<b>3. Instructions For The Rest Of Us</b> .....	<b>3</b>
3.1. Installation.....	3
3.2. Uninstall.....	4
<b>4. PC<sup>2</sup> Initialization Files</b> .....	<b>5</b>
4.1. sitelist.ini file.....	5
4.2. pc2v8.ini file.....	6
4.3. reject.ini file.....	8
<b>5. PC<sup>2</sup> Startup Procedure</b> .....	<b>9</b>
5.1. Built-in Commands.....	9
5.2. Server Startup.....	10
5.3. Starting Clients.....	11
<b>6. Configuring the Contest in PC<sup>2</sup></b> .....	<b>12</b>
6.1. Administrator Login.....	12
6.2. User Accounts.....	13
6.2.1. Account Creation.....	13
6.2.2. Account Names and Passwords.....	14
6.2.3. Loading Account Data.....	15
6.2.4. Importing ICPC Data.....	16
6.2.5. Reconnecting Account Logins.....	17
6.3. Contest Problems.....	18
6.4. Contest Languages.....	21
6.4.1. Defining a Language.....	21
6.4.2. Command Parameter Substitutions.....	23
6.4.3. Language Definition Examples.....	23
6.4.4. Language Definitions In Multi-Site Contests.....	25
6.5. Options.....	27
6.5.1. General Options.....	27
6.5.2. Balloon Options.....	28
6.6. Sites.....	29

<b>7. Starting the Contest.....</b>	<b>30</b>
7.1. Clock Control .....	30
7.2. Contest Length .....	32
7.3. Multi-Site Clock Control .....	33
7.4. Practice Sessions: Resetting A Contest.....	35
<b>8. Monitoring Contest Status .....</b>	<b>37</b>
8.1. Startup Status .....	37
8.2. The Runs Display.....	38
8.3. Editing Runs.....	39
8.4. Filtering Runs.....	41
8.5. Clarifications .....	42
8.6. Reports .....	42
<b>9. The PC<sup>2</sup> Scoreboard .....</b>	<b>44</b>
9.1. Overview.....	44
9.2. Starting the Scoreboard .....	44
9.3. Scoreboard Updates.....	45
9.4. Scoreboard HTML Files.....	46
9.5. Scoring Groups.....	47
9.6. Scoring Algorithm.....	49
9.7. Export Data File .....	50
<b>10. Loosely – Coupled Multi-Site Contests .....</b>	<b>51</b>
10.1. Overview.....	51
10.2. Master Site.....	52
10.3. Loosely – Coupled Startup .....	52
10.4. Generating Merged Standings .....	53
10.5. Cross-Site Judging.....	54
<b>Appendix A – pc2v8.ini Attributes .....</b>	<b>56</b>
<b>Appendix B – Networking Constraints.....</b>	<b>61</b>
<b>Appendix C – Restarting A Server.....</b>	<b>63</b>
<b>Appendix D – PC<sup>2</sup> Server Command Line.....</b>	<b>65</b>

<b>Appendix E – ICPC Import/Export Interfaces .....</b>	<b>66</b>
<b>Appendix F – Validators .....</b>	<b>70</b>
<b>Appendix G – Language Definitions .....</b>	<b>78</b>
<b>Appendix H – Extending PC<sup>2</sup> .....</b>	<b>83</b>
<b>Appendix I – Frequently Asked Questions .....</b>	<b>84</b>

# 1. Overview

PC<sup>2</sup> is a dynamic, distributed real-time system designed to manage and control Programming Contests. It includes support for multi-site contests, heterogeneous platform operations including mixed Windows and Unix in a single contest, and dynamic real-time updates of contest status and standings to all sites. This manual describes the steps required to install, configure, and run a contest using PC<sup>2</sup>. Further information on PC<sup>2</sup>, including how to obtain a copy of the system, can be found at <http://www.ecs.csus.edu/pc2>.

PC<sup>2</sup> operates using a client-server architecture. Each site in a contest runs a single PC<sup>2</sup> *server*, and also runs multiple PC<sup>2</sup> *clients* which communicate with the site server. Logging into a client using one of several different types of PC<sup>2</sup> accounts (Administrator, Team, Judge, or Scoreboard) enables that client to perform common contest operations associated with the account type, such as contest configuration and control (Administrator), submitting contestant programs (Team), judging submissions (Judge), and maintaining the current contest standings (Scoreboard).

PC<sup>2</sup> clients communicate only with the server at their site, regardless of the number of sites in the contest. In a multi-site contest, site servers communicate not only with their own clients but also with other site servers, in order to keep track of global contest state. The following communication requirements must therefore be met in order to run a contest using PC<sup>2</sup>: (1) a machine running a PC<sup>2</sup> server must be able to communicate via TCP/IP with every machine running a PC<sup>2</sup> client at its site; and (2) in a multi-site contest, every machine running a PC<sup>2</sup> server must be able to communicate via TCP/IP with the machines running PC<sup>2</sup> servers at every other site<sup>1</sup>. In particular, there must not be any “firewalls” which prohibit these communication paths; the system will not operate if this communication is blocked<sup>2</sup>. It is not necessary for client machines to be able to contact machines at other sites.

Each PC<sup>2</sup> module (server or client) reads one or more “.ini” initialization files when it starts; these files are used to configure the module at startup. The client module also tailors its configuration when a user (Team, Judge, etc.) logs in. In a typical PC<sup>2</sup> contest configuration, each Team, Judge, etc. uses a separate physical machine, and each of these machines runs exactly one client module. It is possible to have multiple clients running on the same physical machine, for example by having different users logging in to different accounts on a shared machine. In this case, each user (Team, Judge, etc.) will be executing their own “Java Virtual Machine (JVM)”, and must have their own separate directory structure – including their own separate copy of the PC<sup>2</sup> initialization files in their account.

Setting up and running a contest using PC<sup>2</sup> involves the following steps: (1) installing Java and PC<sup>2</sup> on the contest machines; (2) creating/editing the necessary initialization files; (3) starting the server(s) and clients(s); (4) configuring PC<sup>2</sup> for the contest via an Administrator client; and (5) starting the contest so that users (Teams and Judges) can log in. These steps are listed in checklist form in the next chapter, and are described in detail in the remainder of this manual.

---

<sup>1</sup> However, the system also includes a “loosely-coupled” mode of operation which allows sites in a multi-site contest to communicate “offline” without the need for direct live interconnection. See the section titled “Loosely-Coupled Multi-Site Contests” for additional information.

<sup>2</sup> See the Appendix titled “Networking Constraints” for further details on using PC<sup>2</sup> over networks.

## 2. PC<sup>2</sup> Startup Checklist for Geniuses

For those people who hate to read manuals and would rather take a chance with a shortcut list, here is a *very terse* summary of the steps necessary to install PC<sup>2</sup> and get it running. Please note that this is provided as a convenience for geniuses (or gluttons for punishment). The remainder of the manual was written to help everyone else. If you have problems after following this list, *please read the rest of the manual* before sending us a request for help.

- ❑ Install Java (version 1.3.1 or greater) ;
- ❑ Install PC<sup>2</sup> by unzipping the PC<sup>2</sup> distribution to the PC<sup>2</sup> installation directory;
- ❑ Add the Java *bin* directory and the PC<sup>2</sup> installation directory to the PATH;
- ❑ Add “.”, java/lib, and the PC<sup>2</sup> installation directory to the CLASSPATH;
- ❑ Modify the *sitelist.ini* file as necessary to specify each site server name.
- ❑ Edit the *pc2v8.ini* file to point servers and clients to the server IP:port and to specify the appropriate site server name; put the modified .ini file on every server and client machine;
- ❑ Start a PC<sup>2</sup> server using the command “pc2server” and answer the prompted question.
- ❑ Start a PC<sup>2</sup> Admin client using the command “pc2admin” and login using the name “root” and password “root”.
- ❑ Configure at least the following contest items via the Admin:
  - ❑ Accounts (generate the necessary accounts);
  - ❑ Problems (create one or more contest problems, specifying the problem input data file if there is one);
  - ❑ Languages (create one or more contest languages, specifying the language name, compile command line, executable filename, and program execution command line).
- ❑ Press the “Start Contest” button on the Admin “Time/Reset” tab;
- ❑ Start a PC<sup>2</sup> client on each Team and Judge machine and log in using the Admin-created accounts and passwords.
- ❑ Start a PC<sup>2</sup> client on the Scoreboard machine and log in using the “board1” Scoreboard account/password; arrange for the scoreboard-generated HTML files to be accessible to user’s browsers.

### **3. Instructions For The Rest Of Us**

In the event that the preceding checklist is a bit too terse, the remainder of this manual discusses the details of using PC<sup>2</sup> to configure and run a contest. The first step is to install the necessary software, as described in this chapter. The remaining chapters of the manual cover initialization files, starting the system, configuring the system for a contest, starting the contest, using the PC<sup>2</sup> scoreboard, and special considerations for multi-site contests without an active network connection between sites. In addition several appendices cover details of certain topics.

#### **3.1. Installation**

1. Install the Java Software Development Kit (SDK) or Java Runtime Environment (JRE), version 1.3.1 or later on each machine. The remainder of this manual assumes that “\$JAVAHOME” represents the SDK installation directory. For information on obtaining the Java SDK, visit <http://www.javasoft.com>.
2. Add “\$JAVAHOME/bin” directory to the PATH environment variable on each machine (i.e., for each user).
3. Create a directory on each machine to house the PC<sup>2</sup> system. The remainder of this manual assumes that “\$PC2HOME” represents the PC<sup>2</sup> installation directory.
4. Add “.” (‘the current directory’), “\$JAVAHOME/lib”, and “\$PC2HOME” to the CLASSPATH environment variable on each machine. (Note: the JRE does not use the CLASSPATH variable; when using the JRE instead of SDK it is necessary to use the “-cp” parameter. See the JRE documentation for details.)
5. Download the file **pc2v85.zip** (or gzip) from the PC<sup>2</sup> website into the \$PC2HOME directory (that is, into the directory into which you want PC<sup>2</sup> installed).
6. Unzip the **pc2v85.zip** file, being sure to tell the unzip program to “retain directory hierarchy” and “preserve case sensitivity”. This should create directories named “pc2”, housing the PC<sup>2</sup> system, and “com”, housing the IBM common Java facilities. It should also create several “.ini”, “.bat”, and other files, along with directories named “doc” and “samps”, which contain the system documentation and a variety of sample scripts, files, and other goodies you might want to examine. The file “index.html” can be used to browse the documentation.
7. Add the “\$PC2HOME” directory to the PATH environment variable on each machine (i.e., for each user).

## **3.2. Uninstall**

To uninstall PC<sup>2</sup> it is only necessary to undo the above installation steps; that is, remove the \$PC2HOME directory and its contents and restore the system environment variables to their former values . PC<sup>2</sup> itself does not make any changes to any machine locations outside those listed above either during installation or execution. In particular, for example, it makes no entries in the “registry” in a Windows environment, nor does it copy any files to locations outside the installation directories in any environment.



## 4. PC<sup>2</sup> Initialization Files

When a PC<sup>2</sup> module (server or client) begins running, it reads a set of one or more “initialization files” from the directory in which it was started. The Contest Administrator must ensure these files are present as needed, and edit them as necessary, prior to starting a PC<sup>2</sup> module. This chapter describes the principle initialization files and their contents (note: some default versions of the initialization files are provided with the PC<sup>2</sup> distribution package; these must be edited as necessary). Further descriptions of initialization files and their contents can be found in the Appendices.

### 4.1. sitelist.ini file

This file is used to tell servers the names associated with sites in the contest. It must be present in the startup directory for each server (recall that there is one server per site). Each line in the file gives the name of one site in the contest. Site names are *case-sensitive*.

Edit the file “**sitelist.ini**” in the \$PC2HOME directory on each machine. Delete the default information in the file if it is there, and replace it with one line for each site in the contest, giving each Site Name (one site per line).

- NOTE: *it is VERY IMPORTANT that ALL SITES IN THE CONTEST HAVE IDENTICAL **sitelist.ini** FILE CONTENTS -- i.e. the sites must be listed in the same order in the **sitelist.ini** file at all sites, and the spelling and spacing of site names, including any embedded blanks, must be the same in all **sitelist.ini** files.*
- To avoid errors due to mismatched **sitelist.ini** files, it might be desirable to have the Head Judge or Contest Administrator create one **sitelist.ini** file and transmit identical copies to each of the separate Contest Sites (taking into account different line terminators for different OS environments).

## 4.2. pc2v8.ini file

Every PC<sup>2</sup> module reads a file named “**pc2v8.ini**” at startup. This file provides key initialization information to the module. The file is formatted in sections. Each section starts with a section-name in square brackets; for example, **[server]**. Each PC<sup>2</sup> module reads the entire file, but silently ignores any information which does not pertain to it. All lines starting with “#” or “;” are comments and are also ignored, as are blank lines. The following different section names are recognized: **[server]**, **[client]**, **[admin]**, **[judge]**, **[team]**, and **[board]**.

Each section is made up of lines containing "attribute assignment" statements of the form

**attributeName=value**

The “**attributeName**” is a predefined string chosen from list of PC<sup>2</sup> configuration attributes. The “**value**” is the value to which the corresponding attribute is set when the **pc2v8.ini** file is read by a module. No spaces are allowed in front of the “value” after the equal-sign.

Some attribute assignment statements are specific to particular sections and have no meaning for other sections (or for the modules that read them). Other attribute assignments are relevant to multiple sections/modules and can appear in different sections.

A complete list of the predefined attributes is given in the Appendices. Most attribute assignments are optional and default values are used if an attribute assignment is not present in the **pc2v8.ini** file. However, certain attributes *must* be specified in the **pc2v8.ini** file in order for the system to function properly.

Specifically, on a server machine the **[server]** section must contain an attribute assignment giving to the server the name of the site it is serving, using a name specified in the **sitelist.ini** file. Also, the **[client]** section on a client machine (Administrator, Team, Judge, or Scoreboard) must contain an attribute assignment giving the name of the site (from the **sitelist.ini** file) to which it belongs, and also must contain an attribute assignment giving the IP address and port number at which it should contact its server. (Note: PC<sup>2</sup> servers expect by default to be contacted initially on port 50002; see the appendices for further details on port number usage in PC<sup>2</sup>.)

Based on these rules, a minimum sample **pc2v8.ini** file is shown below. Note that since all modules ignore sections and attributes which do not apply to them, it is permissible to create a *single* **pc2v8.ini** file containing all the required entries and put this same file on all machines at a given site.

```
# sample pc2v8.ini file for site named "Site1"

[client]
# tell the client what site it belongs to
# and where to find its server (IP and port)
site=Site1
server=192.168.1.117:50002

[server]
# tell the server which site it is serving
site=Site1
```

The sample `pc2v8.ini` file shown above would be appropriate for both client and server machines in a single-site contest (except that the IP address in the `[client]` section should be replaced by the actual IP address of the site server, and the site name should be changed if necessary to match the site name given in the `sitelist.ini` file). Alternatively, a file containing just the `[client]` section shown in the sample could be put on all the client machines, and a different file containing just the `[server]` section shown in the sample could be put on the server machine.

For a multi-site contest, one additional requirement exists for the `pc2v8.ini` file. In a multi-site contest, the server at one of the sites is designated the “primary server” and servers at all other sites are designated “secondary servers”. When a primary server is started, it waits for other servers to contact it. When a secondary server is started, it automatically attempts to contact the primary server; this is how the inter-server communication in a multi-site contest is established. (The distinction between whether a server waits to be contacted (is a primary) or initiates remote contact (is a secondary) is in fact the only distinction between “primary” and “secondary” servers.)

In a multi-site contest exactly one of the servers should be started as a primary server (it does not matter which site has the primary server; once communication is established all sites run as peers). The servers at all other sites should be started as secondary servers. Designation of a server as primary or secondary is controlled by the contents of the `pc2v8.ini` file.

By default (that is, in the absence of any information in the `pc2v8.ini` to the contrary), a server assumes it is a primary server when it starts. Designating a server as a secondary server is accomplished by providing an additional entry in the `[server]` section of the `pc2v8.ini` file of the server. This additional entry, known as the *remoteServer attribute*, tells the secondary server the IP address and port number at which it should attempt to contact the primary server. If this `remoteServer` attribute is *not* present in the `pc2v8.ini` file when a server starts, the server implicitly assumes it is a primary server.

Thus for example, the `pc2v8.ini` file on the primary server in a multi-site server might look like the entry in the sample above (since it does not contain any `remoteServer` attribute), whereas the `pc2v8.ini` file on machines at a second site might look like:

```
# sample pc2v8.ini for a second site

[client]
# tell the client what site it belongs to
# and where to find its server (IP and port)
site=Site2
server=192.168.1.104:50002

[server]
# tell the server which site it is serving
# and how to contact the primary server
site=Site2
remoteServer=192.168.1.117:50002
```

Note that the (sample) IP address given in the `[client]` section of this `pc2v8.ini` file for Site2 (a secondary site) is the (hypothetical) IP address of the server for that site, whereas the IP address given in the `remoteServer` attribute in the `[server]` section is the address of the primary

server – the address which the (secondary) server at Site2 should use to contact the primary server and “join the contest”.

### **4.3. reject.ini file**

When a server at a site is started, it reads a file name “**reject.ini**” which contains the “reject messages” which can be sent from the Judges back to Teams when the Team submits a program which is not a correct solution. The Contest Administrator must edit the file **reject.ini** in the \$PC2HOME directory on the server machine so that it contains one line of text for each “NO” message to be sent from the Judges to Teams when a run is rejected.

For example, if the Contest rules specify that the Judges will return one of the three messages “Runtime Error”, “Time Limit Exceeded”, or “Wrong Answer” for failed runs, put each of these messages on a separate line in the **reject.ini** file. Note that the **reject.ini** file must exist and be properly edited *prior to starting the server* at a given site.

The contents of the **reject.ini** file thus determines the set of choices from which a Judge can choose when responding to a team’s submission. PC<sup>2</sup> automatically provides the Judge with a choice of “**Yes**” (this is a correct solution to the problem), plus each of the messages listed in the **reject.ini** file. PC<sup>2</sup> automatically prepends the characters “**No –**” to each “reject” message sent to a team when a Judge determines that the submission is not a correct solution to the problem. The text strings in the **reject.ini** file should contain only the distinguishing portion of the message.

The Contest Administrator should insure that every site in a multi-site contest has the same set of reject message strings in the **reject.ini** file at the site.

## 5. PC<sup>2</sup> Startup Procedure

### 5.1. Built-in Commands

Once PC<sup>2</sup> has been installed and the necessary “.ini” files have been properly set up (edited), the normal PC<sup>2</sup> startup procedure is to start a primary server, then start an Admin client connected to that server and use the Admin client to configure the contest details (problems, languages, etc.). Once the contest has been fully configured using the Admin client, secondary servers can be started at remote sites (if any), followed by additional clients at both the primary and (in the case of a multi-site contest) secondary sites.

The PC<sup>2</sup> distribution comes with a collection of “command scripts” designed to simplify starting the various modules. The available command scripts and their corresponding functions are listed below.<sup>3</sup> To invoke the specified function, simply type the corresponding command at a command prompt while in the \$PC2HOME directory.

Command	Function
<b>pc2reset</b>	Creates a backup zipped archive of all existing PC <sup>2</sup> databases, then clears all databases to the state matching a fresh installation <sup>4</sup> .
<b>pc2server</b>	Starts a PC <sup>2</sup> Server
<b>pc2admin</b>	Starts a PC <sup>2</sup> Client expecting an Administrator login
<b>pc2team</b>	Starts a PC <sup>2</sup> Client expecting a Team login
<b>pc2judge</b>	Starts a PC <sup>2</sup> Client expecting a Judge login
<b>pc2board</b>	Starts a PC <sup>2</sup> Client expecting a Scoreboard login

Thus the normal startup procedure<sup>5</sup> would be to type the command “**pc2server**” to start a server, then in a separate command window to type the command “**pc2admin**” to start an Administrative client to be used to configure the contest details in PC<sup>2</sup>. Once the contest details are configured, other clients can be started (as well as servers at other sites) using the appropriate commands.

---

<sup>3</sup> In a Windows environment the command scripts are all “batch files” whose names correspond to the given command name followed by “.bat”. In a Unix environment the command scripts are all Bourne Shell scripts whose names match the given command names. Thus the same command name can be used regardless of the underlying OS.

<sup>4</sup> Except that system “.ini” files are not restored; any changes made to PC<sup>2</sup> “.ini” files will be retained even after executing a **pc2reset** command. The zip archive created by the **pc2reset** command can be used to completely restore the state of a prior contest (or practice session), simply by unzipping the archive into the \$PC2HOME directory (archives are stored in a directory named “archive” within the \$PC2HOME directory).

<sup>5</sup> If there is still PC<sup>2</sup> data present from a previous contest, e.g. from a practice session, the normal startup procedure is to first type the “**pc2reset**” command

## 5.2. Server Startup

When a server is started (using the command “`pc2server`”), the user will be prompted with a question, as follows:

**Are there already servers up (Y, N, or Q) ?**

The server asks this question to help it determine whether this is a normal contest startup, or whether instead this is a server which is being “restarted” following a crash. The following describes how the question should be answered.

For a single-site contest, the answer should always be “N”, since there should never be any other servers running (the Contest Administrator must ensure that any previously-running PC<sup>2</sup> servers – e.g. from a practice session – have been shut down prior to starting a server).

For a multi-site contest:

- If this is the first startup of the primary server, the answer should be “N” since there should not be any other servers running. In this case the server will simply wait to be contacted by other (secondary) servers.
- If this is the first startup of a secondary server, the answer should be “Y” since the primary server should already have been started. In this case the (secondary) server will use its `pc2v8.ini` information to contact the primary server.
- If this is a *restart* of a server which *crashed* or was shut down for some reason during a contest (thus this is a case of a server attempting to *rejoin* a contest which is already in progress), the answer should be “Y” since there should still be other site servers running.<sup>6</sup> In this case additional considerations apply; refer to the Appendix “Restarting a Server” for further details.

Once a server has been started according to the preceding steps, the server displays a series of log messages on the screen. If the `pc2v8.ini` file indicates this is a secondary server (i.e., it has a “`remoteServer=xyz`” entry) the server will attempt to connect to the primary server (as specified by “`xyz`”). In either case, when the server is fully initialized it will display a message similar to:

```
Fri May 09 19:34:52 PDT 2003 main Server at site "xxxxxxx" is ready. (id=1)
```

where “xxxxxxx” is the name of the site as it appears in the `sitelist.ini` file. If any errors occur or the server fails to produce the “server is ready” message, check the files “`pc2.log`” and “`pc2serv.log`” for further details.

---

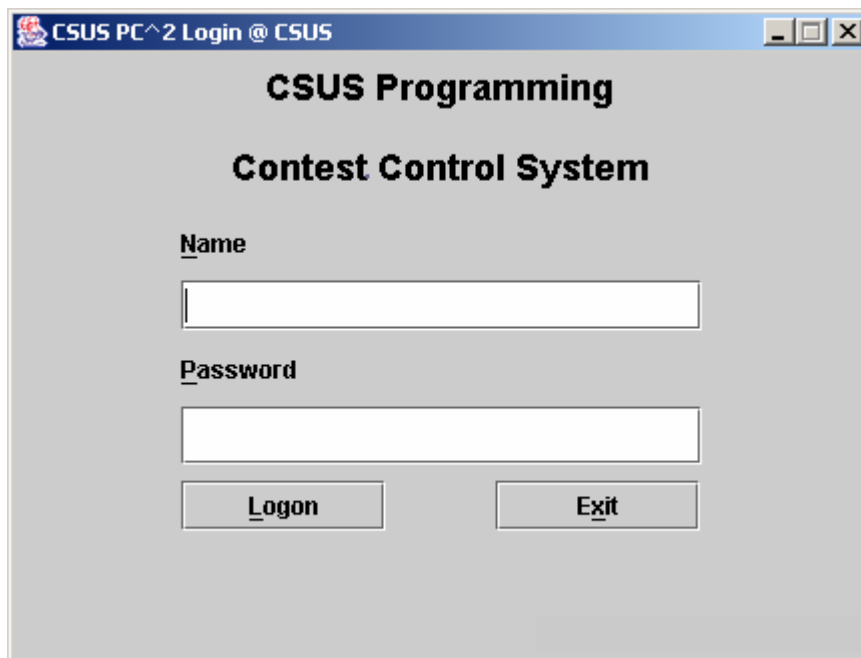
<sup>6</sup> In the event that *all* contest servers crashed or were shut down for some reason, then the restart procedure is simply to begin over with the “first startup” procedure by restarting first the primary server, then the secondary servers.

Once a server is running it can be stopped simply by terminating the Java VM under which it is running. This is typically done, for example, by typing ^C (control-C) at the command window from which the server was started. Note that stopping a server will *immediately* terminate all PC<sup>2</sup> activities at that site.

### 5.3. Starting Clients

Once a PC<sup>2</sup> server is running at a site, users (Contest Administrators, Judges, and Teams) at the site can start PC<sup>2</sup> clients to login and use the system<sup>7</sup>. The normal procedure is first to start a client using the “`pc2admin`” command and login as the “root” administrator in order to configure the contest. Subsequently each Contest Judge would start a client using the “`pc2judge`” command, and each Team would start a client using the “`pc2team`” command. The Contest Administrator would normally also start a PC<sup>2</sup> scoreboard using the “`pc2board`” command, logging in using the PC<sup>2</sup> account “board1”.

Each time a client is started, the client will read its `pc2v8.ini` file to determine its site name and the location of its server, and then contact the server. Following this initialization sequence, the client will display a “login” window as shown below, indicating that it is ready to accept a user (Team, Judge, Administrator, or Scoreboard) login. Depending on the logging levels specified in the client’s `pc2v8.ini` file, the progress of these steps will be displayed and/or written to the file `pc2.log` in the client’s startup directory. If any errors occur or the client fails to produce the login screen, check the “`pc2.log`” file for details.



---

<sup>7</sup> In the case of users logging in to a server (e.g. via Xterminals under Unix) rather than where each user has their own machine, each user must start a client on the server via their terminal window. Each client must be started in its own separate directory, which must contain the appropriate initialization files. Under Xwindows, the DISPLAY environment variable can be used to direct PC<sup>2</sup> graphical output from the client back to the Xterminal.

## 6. Configuring the Contest in PC<sup>2</sup>

### 6.1. Administrator Login

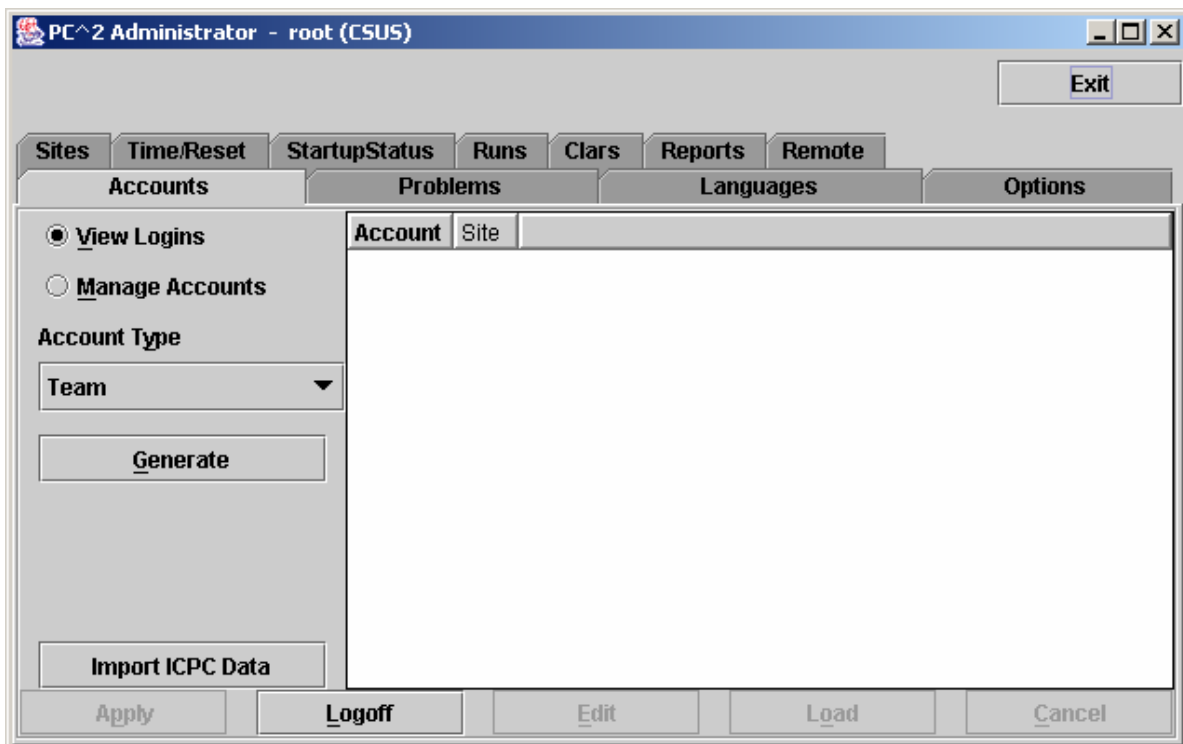
Once PC<sup>2</sup> is set up and running, it is necessary to “log in” to the system via a client login window in order to use the system. A PC<sup>2</sup> “account” is required in order to log in. Initially only a *single* account exists; the account name (“login ID”) of this single account is “**root**”. This account is a *master Administrative account* which is used to configure the PC<sup>2</sup> system initially for the contest. Regular users (especially Teams) should NOT be given access to this account.

The default password for the **root** account is **root** (same as the login name). Note that the default master password is given right here in this paragraph of this document, which is publicly available on the Web.

***Caveat Administrator : change the root password!***

Passwords can be changed via the “Manage Accounts” function on the “Accounts” tab; see below.

After logging in to the “**root**” Administrator account, the following screen will be displayed. This is referred to as the “main Administrator screen”. It provides a series of “tabs” across the top to select various contest administration functions. The tabs which are used to configure the contest prior to starting are described in the remainder of this chapter. Tabs used to start the contest and monitor its progress are covered in the following chapters.

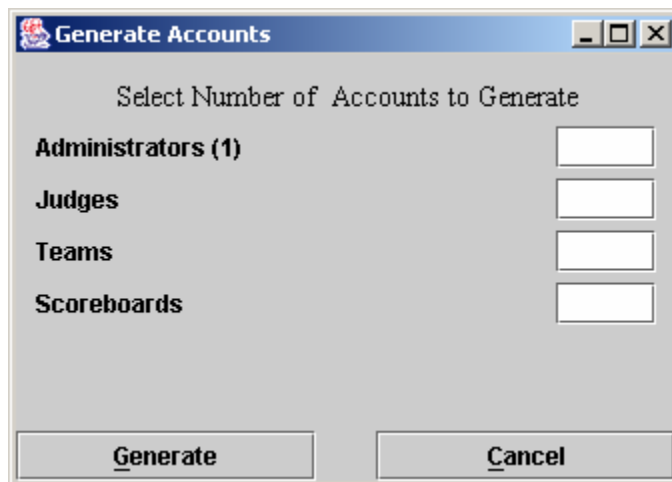




## 6.2. User Accounts

### 6.2.1. Account Creation

Before any logins other than **root** can occur, it is necessary to create user accounts. To create accounts for users, click the **Generate** button on the **Accounts** tab on the main Administrator screen. This will display the following screen:



The screenshot shows a window titled "Generate Accounts" with a standard Windows-style title bar. The main content area contains the text "Select Number of Accounts to Generate" followed by four rows of labels and input boxes: "Administrators (1)", "Judges", "Teams", and "Scoreboards". Each label is followed by an empty text input box. At the bottom of the window are two buttons: "Generate" and "Cancel".

Note that the **(1)** to the right of the “Administrators” label means that currently one Administrator account exists – that one account is the **root** account – and no other accounts exist. It is necessary to create one Team account for each team at this site, one Judge account for each person who will be judging the contest at this site, and at least one Scoreboard (“board”) account if the PC<sup>2</sup> scoreboard is going to be used at this site for tracking contest results. (It does not hurt to generate a few extra accounts in each category, for flexibility.)

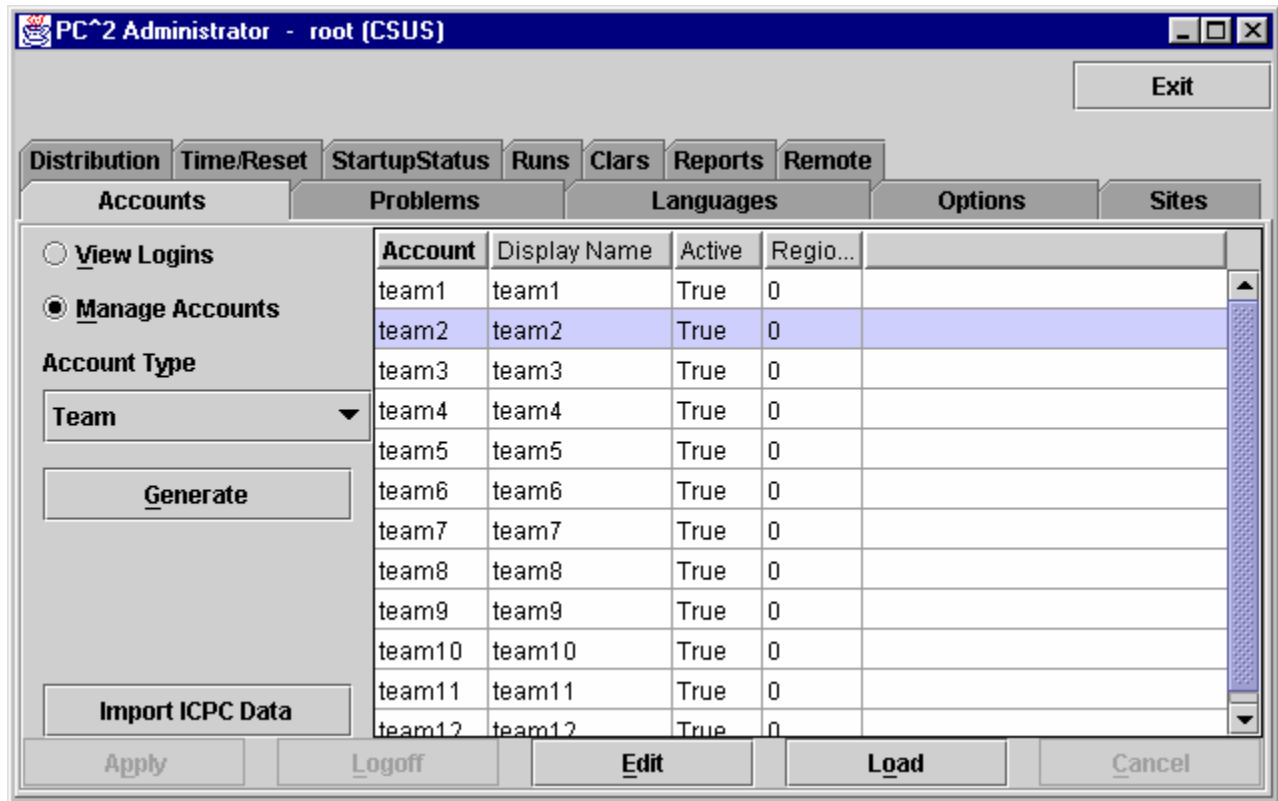
Enter the desired number in each box, then click the **Generate** button. Depending on the number of accounts, number of sites, and communication delays, it will take anywhere from a few seconds to several minutes for the account generation to complete. Once the accounts are generated the system will automatically return to the main Administrator screen. PC<sup>2</sup> accounts always start with the word **team**, **judge**, **admin**, or **board**, followed by a number. See the next section for information on viewing/changing the state of generated accounts.

Accounts in PC<sup>2</sup> are *site-specific*. This means that in a multi-site contest, an Administrator at *each site* must login as root and create the accounts for that site. (It is possible to have a single person handle account creation for all sites in a multi-site contest, by starting separate Administrator clients which have server IP addresses in their **pc2v8.ini** files pointing to different site servers. In any case, however, it is necessary to start a separate Administrator client connected to *each* site server and create the accounts to be used at that site; there is no way in the current version of PC<sup>2</sup> to perform multi-site account creation using just a single Administrator client.)

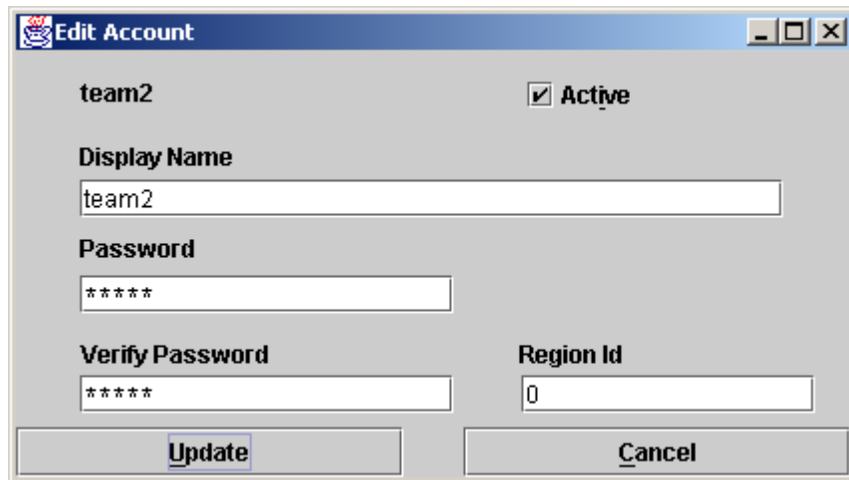
## 6.2.2. Account Names and Passwords

Each generated account will be created with a password, but at present the default (only available) specification for passwords on newly-generated accounts is “Passwords same as Account Name”. This means for example that the password for the account “team1” is “team1”; the password for the account “judge1” is “judge1”; etc. Each account name and password are created with all letters in lowercase.

Passwords for accounts can be changed from their default values by editing each account. To do this, click on the “**Manage Accounts**” radio button on the **Accounts** tab of the main Administrator screen, then use the “**Account Type**” drop-down list to select the type of account you wish to manage (change). For example, the following screen displays the list of Team accounts.



To edit an account, click on the account in the display grid to select it (“team2” has been selected in the display shown above), and then click the “Edit” button. This will display a new “**Edit Account**” window, shown below:



The “Display Name” for an account is the name which will appear on the PC<sup>2</sup> Scoreboard; this can be set to any desired value (such as the name of the team’s school, or the team member’s names). The “Password” and “Verify Password” fields can be used to set any desired password for the account.

The “Active” checkbox determines whether a team account will be considered in computing the scoreboard standings; if there are some team accounts which will not be used then you should uncheck their “Active” checkboxes – otherwise they will appear on the scoreboard as teams which have solved no problems. *Note that the “Active” checkbox only determines whether a team appears on the scoreboard*; teams which are not “Active” can still log in, submit runs, and otherwise participate in the contest. This is designed to allow “guest” or other “non-competitive” teams to participate. (To prohibit *any* activity from a team account, change the account password.)

The “Region ID” field is used to associate accounts with different “regions” or “groups”. This is used in conjunction with the PC<sup>2</sup> scoreboard for displaying rankings of different subgroups (see the chapter on the PC<sup>2</sup> Scoreboard for further details).

Note that PC<sup>2</sup> accounts are unrelated to any user accounts which may otherwise exist on the systems being used for the contest (for example, user accounts provided by the operating system).

In a multi-site contest, newly created PC<sup>2</sup> accounts are automatically distributed throughout the entire system, including across multiple remote sites. As previously noted, accounts are “site-specific”. Note also, however, that accounts at different sites are numbered using the same sequence; the first team account at Site 1 is called “team1”, and the first team account at Site 2 is *also* called “team1”, etc.. Accounts are therefore identified by always giving both the Site number and the Team number, as in “Site1Team1”, which is a *different account* from “Site2Team1”.

### 6.2.3. Loading Account Data

Since editing account data (e.g. Display Names, Passwords, etc.) interactively for every account is cumbersome, it is desirable to be able to prepare the data “offline” ahead of time and

then load it into PC<sup>2</sup>. This can be done by preparing an “account data” file and using the “LOAD” button on the “Manage Accounts” screen to load the data into the system.

An “account data load file” consists of a series of text lines, one for each account to be configured. If it is desired to load data for different types of accounts (teams and judges, for example), it is necessary to prepare a separate account data file for each *type* of account. This is necessary because the LOAD button applies the loaded data to the “currently selected” type of account as shown in the “Account Type” dropdown list on the “Manage Accounts” screen.

The format of the account data load file is as follows. File lines starting with ! or # in the first column are ignored. Each non-comment line has FOUR fields, separated by a pipe (|). The fields are:

```
Account_number
Display_name
Active (either "true" or "false")
Password
```

For example, to initialize accounts “team1”, “team2”, and “team3” so that the “team1” account displays on the scoreboard with the name “Number 1” and has a password of “pass1”, while the “team2” account displays on the scoreboard with the name “Team Number 2” and has a password of “myPass”, and the “team3” account is made inactive (does not display on the scoreboard), the following entries would be placed in the load accounts data file for teams:

```
1|Number 1|true|pass1
2|Team Number 2|true|myPass
3|My School Name|false|
```

Empty (or blank) Display Name and/or password fields cause the corresponding item to be set to the empty string. An empty (or blank) "active" field is considered **false**. Imported values overwrite any values that were in the system previously. Also, it is not necessary to provide a record in the data file for every account; the Account Number field determines which accounts will be modified (all unlisted accounts will remain unchanged).

Note that there is no way in the current version of PC<sup>2</sup> to assign a “Region ID” to an account via a “Load” operation; if regional groupings are being used then the Region ID values must be assigned manually (see the chapter on the PC<sup>2</sup> scoreboard for further details).

#### 6.2.4. Importing ICPC Data

PC<sup>2</sup> was designed for supporting the ACM International Collegiate Programming Contest, including its local and Regional contests worldwide. The ICPC maintains an online Contest Registration system which is used by Regional Contest Directors (RCDs) around the world to manage participation in the various ICPC Regional Contests.<sup>8</sup> PC<sup>2</sup> provides interfaces to import contest registration data from the ICPC Registration system, and also to export contest results back to the ICPC web site. See the Appendix on ICPC Import/Export Interfaces for further information on importing/exporting ICPC Registration system contest data.

---

<sup>8</sup> Visit the ICPC web site at <http://icpc.baylor.edu/icpc/> for further details.

### 6.2.5. Reconnecting Account Logins

During a contest it is possible (likely, it seems from experience) that a team or judge will somehow manage to kill their PC<sup>2</sup> Java VM after having logged in to PC<sup>2</sup>. This can happen in a variety of ways, such as a total machine crash or a simple inadvertent command to “kill” the “command window” which is running the Java VM (users should be warned not to terminate this command window).

In the event that a user becomes disconnected and/or kills PC<sup>2</sup> somehow, they will (eventually) wish to log back in to the system. However, in the current version of PC<sup>2</sup> the server has no way of detecting the loss of connectivity to the client machine; the server therefore will think they are still “logged in”.<sup>9</sup> For security (to prevent teams from logging in and acting as another team) PC<sup>2</sup> refuses to allow multiple logins of the same account. Thus a disconnected user may not be able to log back in (because the server thinks they are “already logged in”).

As a compromise in this situation, PC<sup>2</sup> uses the following rule when it receives a login request from a client: if there is *already* a registered client login using that account, then *if* the IP address of the machine requesting the new login is identical to the IP address of the currently registered login for that account, then the server attempts to communicate with the already-registered client (in a sense it ‘pings’ the client). If this communication *fails*, PC<sup>2</sup> assumes this is a case of a client which became “disconnected” somehow, and it *discards* the old login registration and accepts the new registration instead.<sup>10</sup> This procedure effectively means that most “disconnections” can be resolved simply by having the team (or judge) log back in (this was not true in earlier versions of PC<sup>2</sup>).

If for some reason a client was logged in, got disconnected, and had to change to a machine with a different IP address, the server will not recognize this as a reconnection login, and it will enforce the rule prohibiting multiple logins of the same account (since it will still have the old login record). To solve this, the **Accounts** tab on the main Administrator screen provides a function to “force a logout” of a currently logged-in account.

To invoke the “forced logout” function, click on the “**View Logins**” button on the **Accounts** tab, then click on the row in the display showing the user account that needs to be “logged out”. This will activate the “**Logout**” button (see the **Accounts** tab screen, above). Pressing this button will then force the Server to drop the login connection, allowing that account to re-login, whether from the same IP address or not.

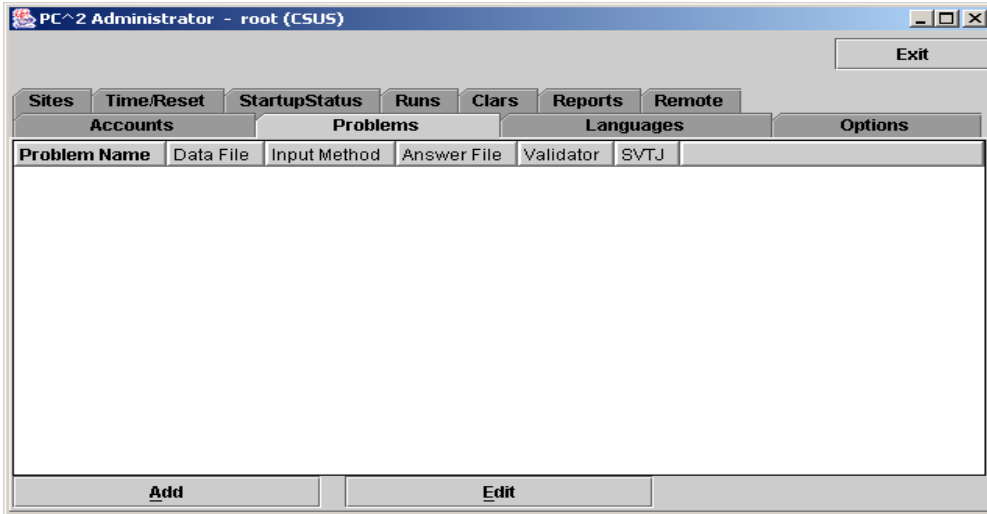
---

<sup>9</sup> Yes, this is on our list for improvements in “future versions”...

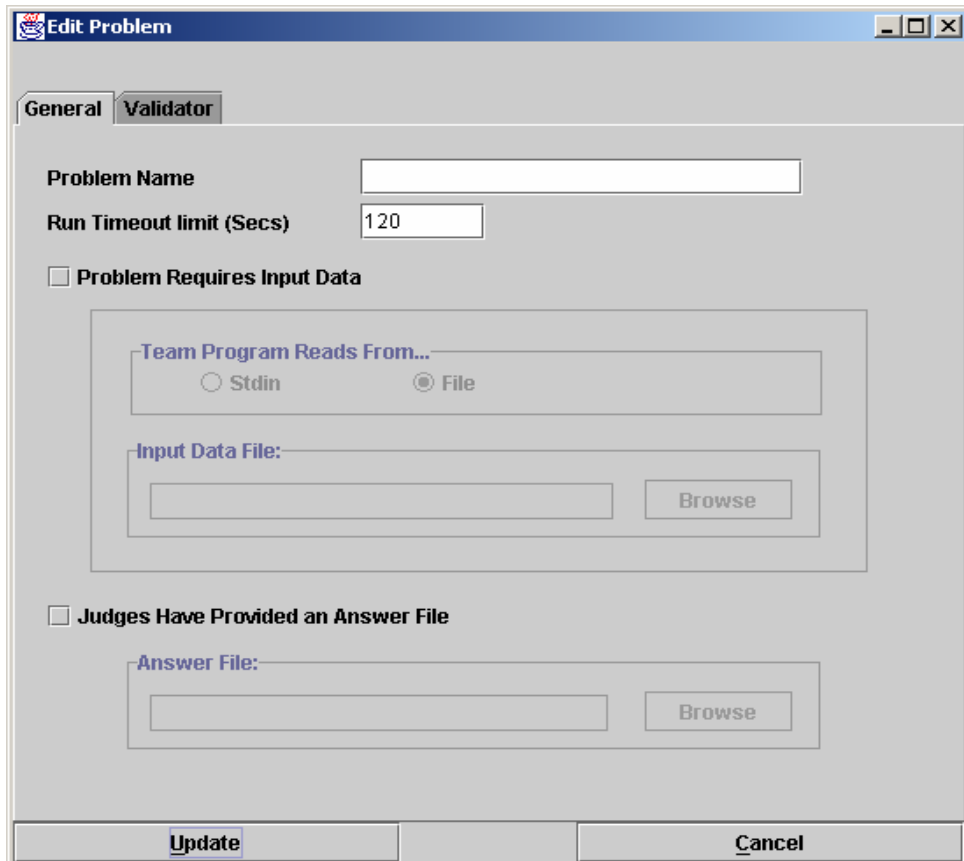
<sup>10</sup> If the communication with the original client *succeeds*, PC<sup>2</sup> assumes that the new login request is an attempt at a multiple login using the same account, which it rejects by sending an “account already logged in” message back to the new client.

### 6.3. Contest Problems

PC<sup>2</sup> must be provided with information about the problem set to be used in the contest. To enter this information, click on the **Problems** tab at the top of the main Administrator screen. This will produce a display similar to the following:



Note that initially no problems are listed since none have been added to the system. To add a problem, click the **Add** button. This will produce the “**Edit Problem**” dialog shown below.



To define a contest problem to the system, perform the following steps using the **Edit Problem** dialog:

- Enter the problem name in the top textbox.
- If the problem requires an input data set, click the “Problem Requires Input Data” checkbox and then
  1. select either “Stdin” or “File”, depending on whether the problem description tells teams to write their programs to obtain input data from “standard input” or from a file<sup>11</sup>, then
  2. use the Browse button to select the data file.<sup>12</sup>
- If the Judges have provided an “Answer File” (a file showing the expected output of a program correctly solving this problem), click the “Judges Have Provided an Answer File” button and then use the Browse button to select the Answer File.
- Click the **Update** button to store the problem information.

As each set of contest problem information is entered, it will be displayed on the main Administrator screen (when the **Problems** tab has been selected). To change some previously entered information for a problem, click on the problem row in the main display to select it, then click the **Edit** button. This will return to the **Edit Problem** dialog, where changes can be made.

The following additional notes apply when entering data using the **Edit Problem** dialog:

- The Run Timeout Limit value (shown as 120 in the sample screen above) is not enforced by PC<sup>2</sup>, in the sense that the system does not automatically stop a submission from executing when the time limit is reached. Rather, a count-up timer is displayed during program execution so that the Judge can tell how long the program has been executing, and the counter value turns **RED** when the specified timeout limit is reached. The timer also includes a button to allow the Judge to terminate the program
- The content of the input data file for a problem is stored internally when the **Update** button is pressed (that is, PC<sup>2</sup> makes an internal copy of the file). For this reason, editing the user’s copy of the file will *not* automatically change the data presented to team programs. To modify the data file for a problem, the contest administrator must **EDIT THE PROBLEM** and load the new copy of the data file, then press the **Update** button again.

---

<sup>11</sup> Note that teams can be instructed to write programs which read from “stdin” even though the administrator provides the input data in a file; PC<sup>2</sup> arranges that the content of the specified file is available in the current directory at runtime (for the case of reading from a file), or that the content of the file is presented to the program’s standard input channel (if that input selection is specified).

<sup>12</sup> In the current version of PC<sup>2</sup>, only one input data set is allowed per problem. To test programs against multiple data sets, place all the data sets in a single file, put a “counter” record at the front specifying the number of data sets, and include instructions in the problem description telling teams to process the input data this way. And yes, it’s another thing on our list of desired improvements for a future version of the system...

- The **Validator** tab on the **Edit Problem** dialog is used to interface an automated judging program for this problem to PC<sup>2</sup>. See the Appendix on Validators for further details.
- All team program output is expected to go to “standard output” (where it is captured by PC<sup>2</sup> and saved for display to the Judges). More specifically, there is no mechanism in the current version of PC<sup>2</sup> for dealing with programs which are written to send their output to a destination other than “stdout” (for example, programs which send their output to a file).<sup>13</sup>
- Contest problems in PC<sup>2</sup> are *global*, in the sense that once a problem definition is entered by the Contest Administrator that problem definition is broadcast to all sites. There is no mechanism for having teams at different sites in a multi-site contest see different descriptions of the same problem. If there is some reason that different descriptions are needed for the same problem (for example, if a problem needs to be described differently at different sites due to OS differences), it is necessary to enter the different problem descriptions effectively as different problems. (While this is not very elegant, it is also something that we *rarely* – virtually never – see in real contests... Said another way, PC<sup>2</sup> views a contest as a set of teams all working on an identical problem set.) Note also that contest problems must appear in the same *order* at each site in a multi-site contest; see the chapter on Loosely-Coupled Multi-Site contests.
- PC<sup>2</sup> copies the input data file for a problem into memory each time a team program for that problem is executed (this is how the architecture manages the insertion of the input data into the input stream of the team program). If the size of the input data set (file) for a problem is particularly large and the system has a relatively small amount of memory, it is possible to exceed the memory limits of the Java Virtual Machine (JVM). This problem can be circumvented by utilizing a script for the “Program Execution Command Line” which copies the required data file into the **\$PC2HOME/execute** directory and then invokes the team program. (See the following section on Contest Languages, and the Appendix on Language Definitions, for further details.)

---

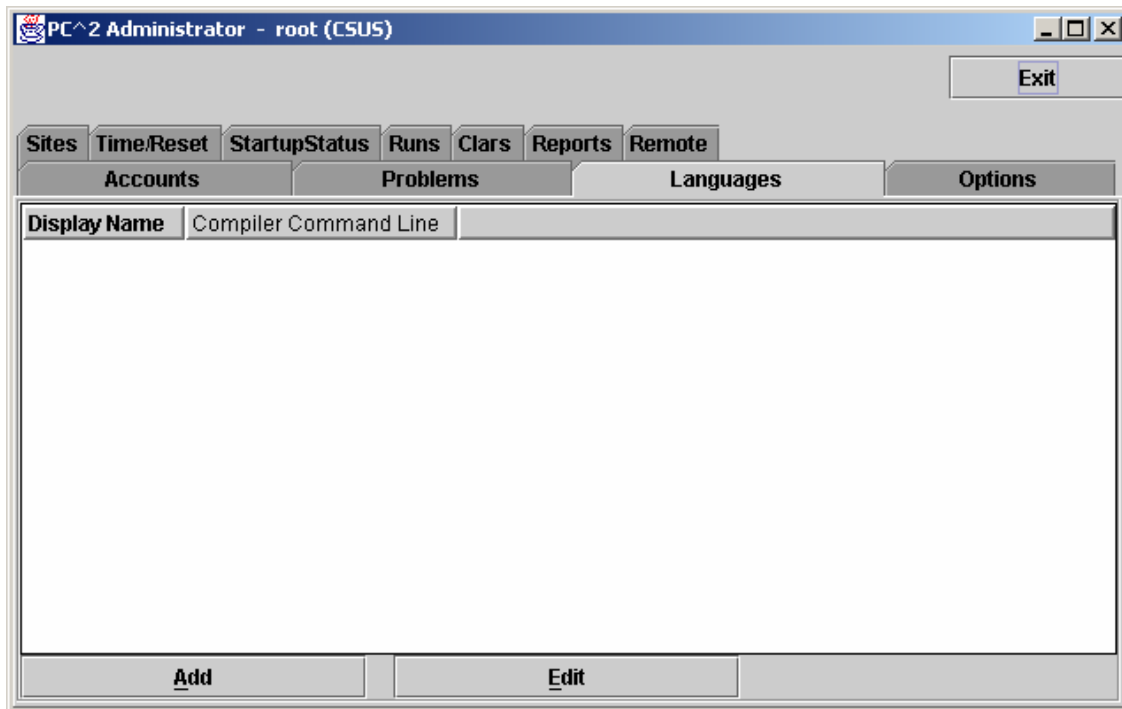
<sup>13</sup> There is in principle no reason a contest administrator could not use the PC<sup>2</sup> “Validator” capability to effectively examine and process output sent to a file by a team’s program – including displaying that output for the Judges. While this is not the primary intent of the Validator capability, it could be used as an effective workaround for this limitation. See the Appendix on Validators for details.



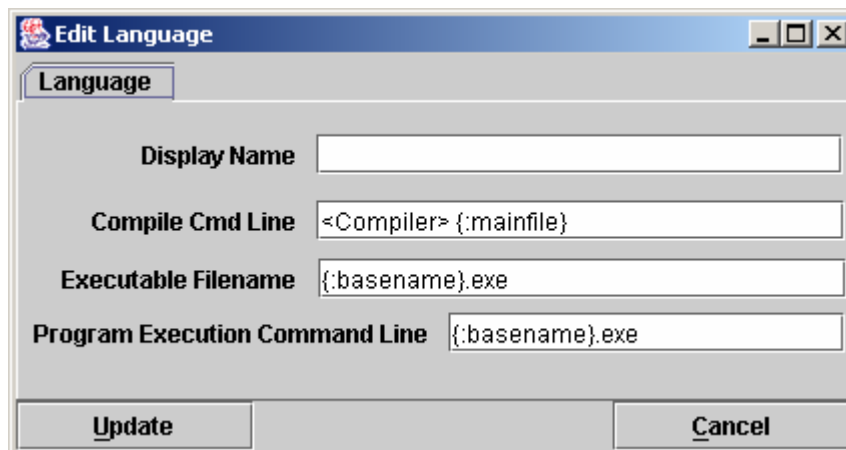
## 6.4. Contest Languages

### 6.4.1. Defining a Language

PC<sup>2</sup> must be provided with information about the programming languages used by contestants (Teams). To enter this information, click the **Languages** tab on main Administrator screen. This will bring up a display similar to the following:



Note that the display is empty because no languages have been defined yet. To add a language description, click the **Add** button. This will bring up an **Edit Language** dialog, similar to the one shown below, containing four fields used to describe the language to PC<sup>2</sup>.



The “Display Name” for a language is the name which Teams will see when they are asked to specify the language in which they have written a program which they are submitting. The Display Name can be any arbitrary text; it does not have to be a real language name (for example, “Local C Compiler” could be a legitimate language Display Name).

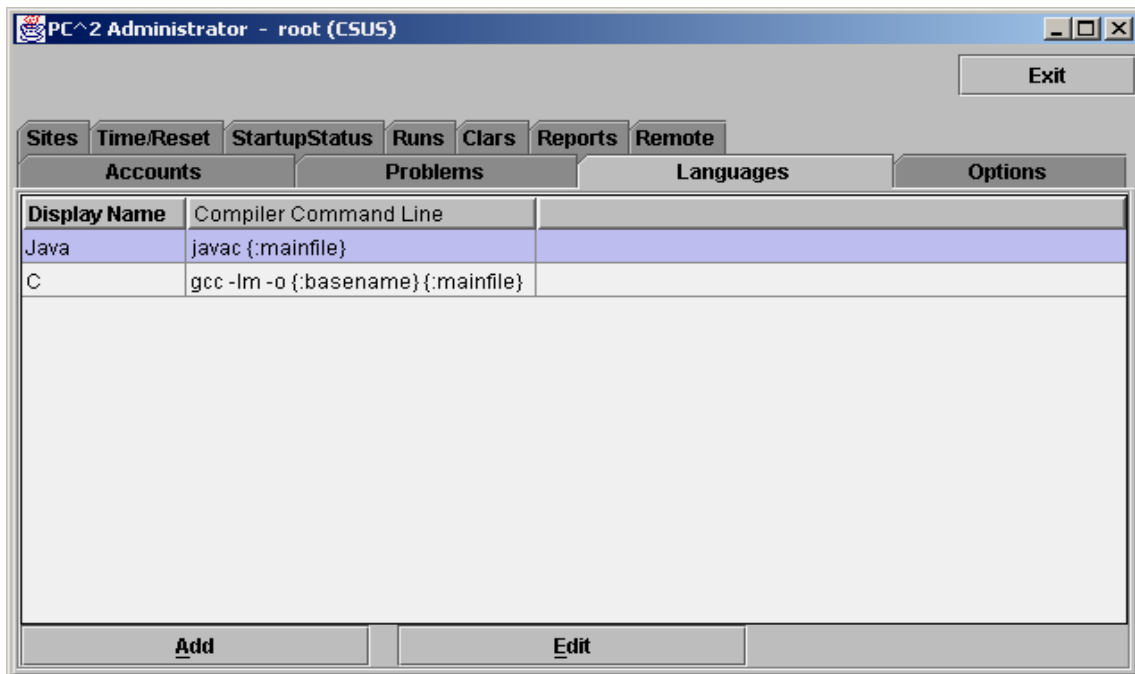
The “Compile Cmd Line” field is used to specify the command line which is used to compile source code and produce an “executable program file” in the language.

The “Executable Filename” field is used to tell PC<sup>2</sup> the name (or more correctly, the form of the name) of the output (executable program) file produced by the compilation process. PC<sup>2</sup> clears its internal execution directory of any instance of the specified executable file prior to compilation, and checks for the existence of the specified executable file following compilation. It interprets the existence of a new executable file as evidence of successful compilation.

The “Program Execution Command Line” field is used to specify the (form of the) command line required to execute (run) the resulting program. Execution is only performed if the preceding steps were successful in producing a new executable file.

The Contest Administrator must define each language to be used in the contest by filling in the four language definition fields, replacing the default values shown above with the appropriate values for the language being defined. (Note that the preceding example screen shows values called “command parameter substitutions” in the language definition fields; see the following sections for further details on the definition fields.)

Once the definitions for a language have been entered, click the **Update** button to store the information and return to the main Administrator screen. The language names will be displayed under the **Languages** tab on the main Administrator screen, as shown below. To add more languages, click the **Add** button again to return to the **Edit Languages** screen. To modify a previously-entered language, click on the row containing the language description to select it and then click the **Edit** button. See the Appendix on Language Definitions for further details.



### 6.4.2. Command Parameter Substitutions

The four language description fields in the **Edit Language** dialog can be “hard-coded” by entering fixed values if desired. For example, the Display Name for a language is normally fixed for the duration of a contest (e.g., “Java”, or “C++”, or “Pascal”).

However, entering fixed values for the Compile Command, Executable Filename, and Program Execution Command fields can be extremely cumbersome and inflexible – the details of these fields may need to change with each different program file submission, for example. In order to provide more flexibility, PC<sup>2</sup> supports the use of “parameter substitutions” in these fields.

PC<sup>2</sup> parameter substitution fields are indicated by matching curly braces, with the first character inside the left curly brace being a colon (‘:’). Following the colon character is exactly one of the predefined PC<sup>2</sup> parameter substitution keywords. Any number of command parameter substitution fields may appear anywhere in a language description field. The currently defined parameter substitution keywords and their corresponding meanings are given below.

<b>Keyword</b>	<b>Meaning</b>
<b>mainfile</b>	Replace with the full name of the submitted file, including any extension (but excluding any ‘path’ specifier on the front of the filename)
<b>basename</b>	Replace with the base component of the file name, omitting any extension (and excluding any ‘path’ specifier on the front of the filename)

The following section shows examples of language definitions, including the use of command parameter substitution fields.

### 6.4.3. Language Definition Examples

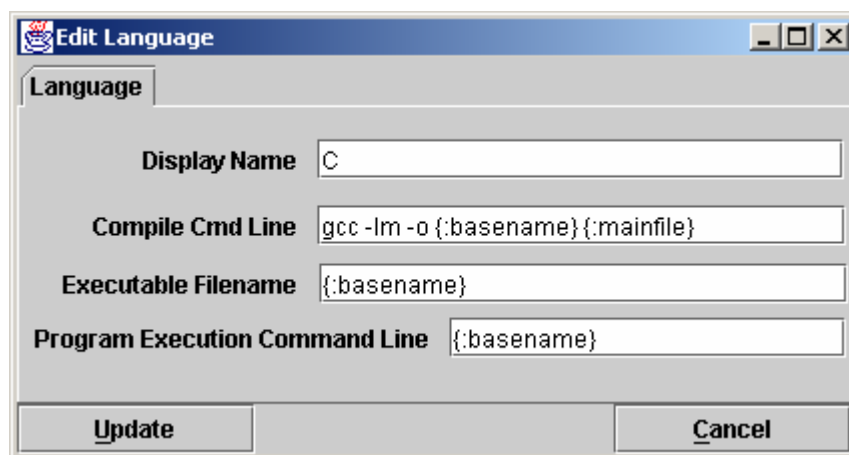
The following screen shows a set of filled-in fields defining a language named “C” and using the GNU GCC compiler.

The compile command line invokes the compiler (“gcc”) and passes it an argument specifying use of the math library (“-lm”). The compile command line also specifies the assignment of a specific name to the “object” (compiled output) file (the “-o” argument), followed by the name to be assigned to the object output file. In this case, the object output file is to have the same name as the base name of the input source code file. (So for example if a team submitted a file named “proga.c”, the object output file would be named simply “proga”, since that is the value to which the “{:basename}” substitution parameter would be expanded.)

The final argument on the compile command line gives the name of the source file to be compiled, which would be expanded from “{:mainfile}” to become “proga.c” if that was the name of the submitted main program source file.

The Executable Filename field indicates that the executable file which produced by the compile command has the same name as the base name of the submitted program; this is because the compile command specifies (via the “-o” argument) that this is the executable file name which should be produced.

The Program Execution command field specifies that the command used to execute the compiled program is simply the same as the name of the executable file produced by the compilation step (and specified in the Executable Filename field), which in this case is again the base name of the original source code file.



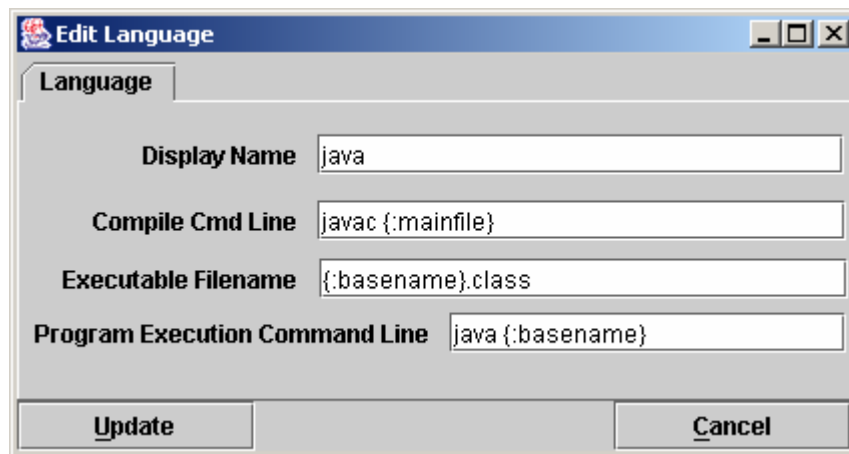
If a team were to submit a C program in a file named **proga.c** using the above language, PC<sup>2</sup> would first execute:

```
gcc -lm -o proga proga.c
```

to compile the program (substituting **proga** for the {:basename} parameter and **proga.c** for the {:mainfile} parameter). It would then check for the existence of an executable file named **proga**, and if that file exists then PC<sup>2</sup> would request the underlying operating system to execute the command:

```
proga
```

The following screen shows a second language definition example: a definition for a language with the display name “java”:



If a team were to submit a Java program in a file named `sumit.java` using this language definition, then PC<sup>2</sup> would execute the following command to compile the program:

```
javac sumit.java
```

Then PC<sup>2</sup> would check for the existence of an executable file named `sumit.class` and if that file exists then PC<sup>2</sup> would execute the following command to run the program:

```
java sumit
```

Note that the form of the language definition fields differs somewhat between the previous example (C) and the second example (Java). This is because of the different ways in which these two languages define the compilation and execution process. Notice also, however, that while the language paradigms are different, the use of command parameter substitutions allows the Contest Administrator easily to provide descriptions of how to handle the differences. The appendices contain further samples of language definitions for specific compilers.

#### 6.4.4. Language Definitions In Multi-Site Contests

Language definitions in PC<sup>2</sup> are *global*. This means that, just as with Contest Problem definitions, when a language definition is entered at one site in a multi-site contest, that language definition will be visible at *all* connected contest sites. However, unlike the situation with Contest Problems (where the problem definitions are usually identical across sites), language definitions may differ between sites – even for the “same language”.

For example, it may be the case that every site allows the use of the “C” language. However, it may also be true that the specific command sequence to invoke the C compiler may differ between sites: a different C compiler might be used at different sites, or even if the same

compiler is used it may be necessary to allow for differences in the “path” needed to access the compiler or for other environmental differences.

One way to deal with differences in language details between sites is to create a different PC<sup>2</sup> language description for each different language/site combination. This can quickly become cumbersome, however; for example, if there are four languages (e.g. C, Java, Pascal, and Perl) and five sites using those languages, it could require entry of up to 20 different language descriptions (Site1C, Site2C... Site1Java, Site2Java,... etc.). This can become particularly unwieldy for Teams, who must search through a list of 20 different languages looking for not just the correct language but the correct language *for their site*.

To avoid this combinatorial explosion of language definitions, a simple technique can be used when defining languages in a multi-site contest: use of *generic language scripts*, tailored at each site for the site-specific configuration.

For example, consider a contest using, say, C, Java, and Pascal. The Contest Administrator should define those three languages in PC<sup>2</sup> using the actual language names (“C”, “Java”, and “Pascal”) as the PC<sup>2</sup> “language Display Names”. However, rather than defining a specific compilation command for each language (which may differ between sites), each language should have as its compilation command a command which invokes a language-specific (but site-independent) *script* (or “batch file”) designed to compile a program in that language.

In other words, for the above three languages, PC<sup>2</sup> language definitions would be created to define the “compilation command” for the language named “C” to be the invocation of a script (batch file) named “*compileC*” (or “*compileC.bat*”); the compilation command for Java would be the invocation of a script named “*compileJava*”; and the compilation command for Pascal would be the invocation of a script named “*compilePascal*”.

Then, at *each site*, the Site Director is responsible for placing on machines at that site a set of scripts or batch files of the corresponding names (e.g. *compileC*, *compileJava*, and *compilePascal*). Within each script at each site is a set of *site-specific commands* which perform the necessary steps (compile a C program, compile a Java program, or compile a Pascal program) in the appropriate site-specific manner.

Note that if necessary, the same technique of “generic scripts” which vary between sites can also be used in specifying the details of “Program Execution Command Line” for languages. That is, the Contest Administrator can specify “*executeC*”, “*executeJava*”, and “*executePascal*” scripts for the program execution language definitions in PC<sup>2</sup> and then arrange for appropriately different script contents at each site.

Note also that PC<sup>2</sup> “command parameter substitutions” may be used in compilation and execution command lines independently of whether the command is invoking a script or not; in this way the Contest Administrator can arrange to pass necessary data (such as the main program file name and/or the base name) to a script.

Using generic script names in PC<sup>2</sup> language definitions and providing site-specific implementations of each language script at each site allows the Contest Administrator to significantly reduce the number of language definitions which teams must deal with, while at the same time retaining the flexibility necessary for dealing with site differences in a multi-site contest.

## 6.5. Options

The **Options** tab on the main Administrator screen displays a screen which allows selection of additional options which can be used to display and/or control various aspects of a contest. These are broken into two sets: **General Options** and **Balloon Options**; each of these categories is listed and described below.

### 6.5.1. General Options

Selecting **General Options** displays a new screen with three tabs: **Judge**, **Team**, and **Scoreboard**. The following tables list the option settings available on each of these screens.

Judge Options	Description
Show Team Numbers to Judges?	Specifies whether or not to reveal the identify of Teams to the Judges while a run is being judged:  Yes – displays the team name/number on each run  No – displays “***” in the place of team names

Team Options	Description
Maximum Output Size	Specifies the maximum amount of output, <b>in Kbytes</b> , which a team program is allowed produce to stdout or stderr. Any output beyond this amount is discarded by the system. The default value is 512 (1/2 MB).
Disable Team’s Viewing Judgements	Specifies the number of elapsed minutes after which judgments are not displayed on team grids. The default value if not specified is 240 (4 hours). <sup>14</sup>

Scoreboard Options	Description
Contest Title	Specifies the contest title which is to be displayed on the scoreboard.
Submission Penalty Points	Specifies the number of penalty points to be added to a team’s score for each “no” judgment for a problem which is successfully solved. The default value is 20 points.

---

<sup>14</sup> Judge clients have a checkbox which allows them to “Send Notification To Team” (or not) when a judgement is rendered. If the Judge checks that box when responding to a run, the Team will still receive a pop-up notification of the judgement regardless of the value of the “Disable Team’s Viewing Judgements” option. This option only sets the time at which judgement results for runs beyond this time are not displayed in the Team’s “run grid”.

## 6.5.2. Balloon Options

In many contests (including the ICPC World Finals), balloons are used to indicate to contestants and spectators alike the general state of the contest. Each time a team solves a problem, a balloon of a specific color is sent to the team and attached on or near their machine. As the contest progresses, the contest floor gradually fills up with a multi-colored display showing how various teams are doing in the contest. (This is a colorful and normally well-received operation; if you have never tried it, we recommend doing so.)

Using balloon notifications in a contest does present some additional management overhead (keeping track of which team should get what color balloon, etc.). Since PC<sup>2</sup> was designed to support the ICPC World Finals (as well as its Regional and Local contests), it contains some built-in support for “balloon operations”. Selecting the **Options** tab on the main Administrator screen and then clicking on “**Balloon Options**” will cause a dialog similar to the one shown below to appear; this dialog is used to configure the handling of balloons in a contest.

Key	Value	Probl...	Color
Email Balloons	<input type="checkbox"/> <b>Enable Emailing</b>	Sum Pro	
SMTP Server			
E-mail address			
Print Balloons	<input type="checkbox"/> <b>Enable Printing</b>		
Print Device(Port)			
Print/Email No's	<input type="checkbox"/> <b>Enable Sending No...</b>		

Balloon options include ability to specify the color of balloon associated with each problem; sending messages to a printer each time a balloon should be delivered to a team; and sending an email message via a specified email (SMTP) server to an arbitrary email account each time a balloon should be delivered to a team. Printed and emailed messages contain the relevant details such as Team, problem, and balloon color. Each of these options can be selected on a per-site basis (so that, for example, sites can use different color balloons for a given problem).

Generation of balloon notifications is handled by the PC<sup>2</sup> Scoreboard machine.<sup>15</sup> Once email and/or printing notification is enabled, every “YES” judgment detected by the PC<sup>2</sup>

<sup>15</sup> More specifically, it is handled by a PC<sup>2</sup> Scoreboard machine logged in under the PC<sup>2</sup> account “board1”; there must be a PC<sup>2</sup> “board1” account running in order to use balloon notifications.



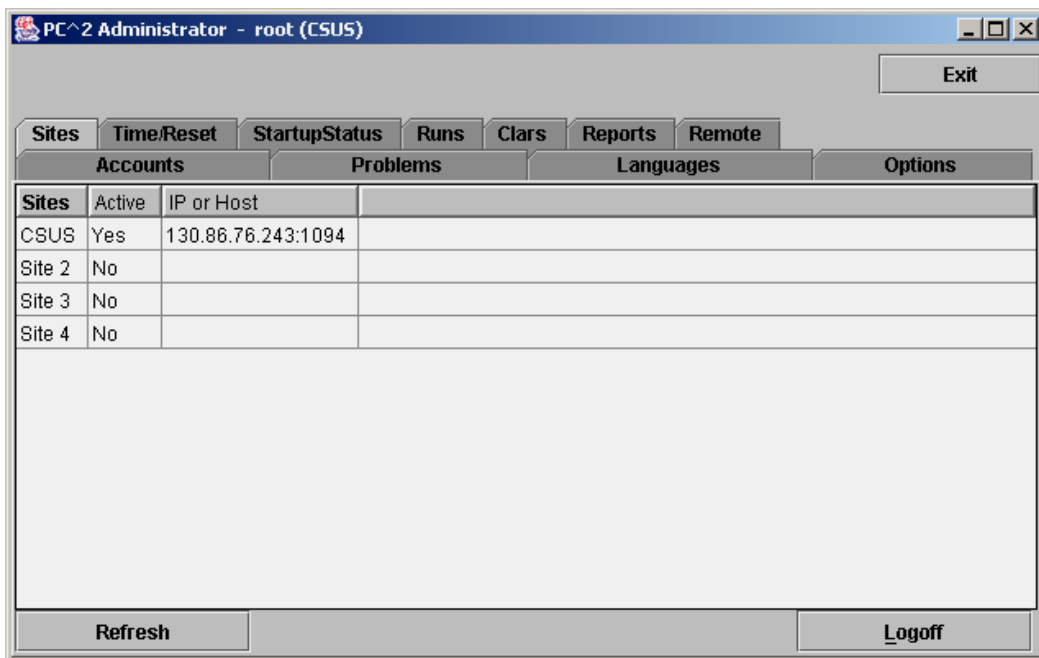
Scoreboard “board1” account will cause an email notification and/or a printed notification to be sent to the configured location.

To enable the use of email balloon notifications, check the “Enable Emailing” box, then enter in the appropriate text boxes the full name of an SMTP email server accessible to the PC<sup>2</sup> Scoreboard machine along with a valid email address.

To enable the use of printed balloon notifications, check the “Enable Printing” box, then enter in the appropriate textbox the device identifier of a printer accessible to the PC<sup>2</sup> Scoreboard machine. If it is also desired to print notifications of “NO” judgments, check the “Enable Sending No’s” box.

## 6.6. Sites

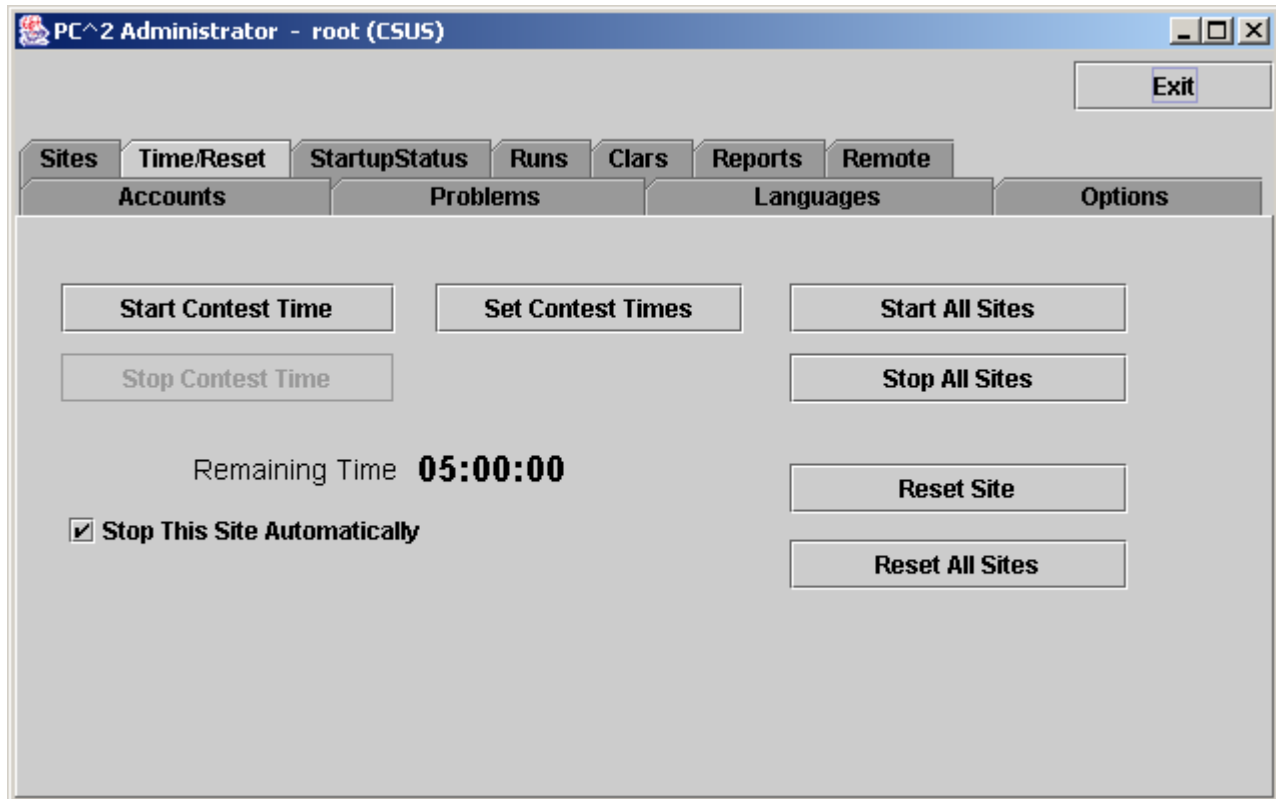
This tab on the Administrator main screen displays a list of all the sites in the contest; this list should be checked to verify that PC<sup>2</sup> knows about all sites. If the site is currently active (connected to the rest of the contest) the IP address for that site’s server is displayed.



## 7. Starting the Contest

### 7.1. Clock Control

Once the contest has been fully configured in PC<sup>2</sup>, the contest clock must be started before teams can submit runs (the Team client will not allow run submissions if the contest clock has not been started). The **Time/Reset** tab on the main Administrator screen is used to control the contest clock display and to start and stop the contest clock. Clicking the **Time/Reset** tab produces a screen similar to the one shown below:



The **Start Contest Time** button is used to tell PC<sup>2</sup> to start the contest clock for the current site (the current site name is shown in parentheses in the title bar). Pressing **Start Contest Time** starts the contest clock running for that site and allows teams at that site to submit runs.

The amount of time remaining in the contest at the current site is displayed in the "Remaining Time" field, and automatically starts counting down as soon as the **Start Contest Time** button is pressed. It continues to automatically update (count down) as long as the contest clock is running.<sup>16</sup>

<sup>16</sup> The remaining time is also displayed on the Team and Judge client screens, and also counts down automatically; however, it is displayed only to a resolution of one minute on those screens.

It is important to note that, from the point of view of PC<sup>2</sup>, the contest does not start until the **Start Contest Time** button is pressed. For this reason **it is important that the Contest Administrator remember to press the Start Contest Time button at the actual time the contest starts.** Failure to do this can produce erroneous scoring results.

Typically, for example, a contest is deemed to have “started” when the contest problems are distributed to the teams. If the PC<sup>2</sup> **Start Contest Time** button is not pressed for another, say, 15 minutes, then that 15 minutes will not be considered to have been part of the contest by PC<sup>2</sup>. If a team were to submit a run 20 minutes after the contest started (i.e. 20 minutes after the problems were handed out), the timestamp on that run would show a contest elapsed time of 5 minutes, not the correct value of 20 minutes. This would produce erroneous values on the scoreboard.

The **Stop Contest Time** button is used to tell PC<sup>2</sup> to stop the contest clock for the current site. The **Stop Contest Time** button can be used to insert a pause in a contest (for example, to allow a break for lunch). During the time the contest is stopped, the contest clock at the site does not count down, and teams are prohibited from submitting runs. When the **Start Contest Time** button is pushed again, the contest clock for the site picks up where it left off.

Note that this means that if a team submits a run one minute before the contest clock is stopped, and then the clock is stopped for 30 minutes of real time, and then the team submits another run immediately after the contest clock is restarted, the timestamps on the runs will be one minute apart. In other words, PC<sup>2</sup> does not consider time during which the contest clock is stopped to be part of the contest. (If this is undesirable – that is, if the Contest Administrator wishes *all* time which elapses to be counted, then simply do not press the “stop” button once the contest has been started. )

PC<sup>2</sup> can be instructed to automatically stop accepting runs from teams at this site for scoring when the contest clock reaches zero (no time remaining). This is done by checking the “**Stop This Site Automatically**” checkbox on the **Time/Reset** screen (the contest must currently be stopped in order to change the state of the checkbox). If this checkbox is checked, then once the contest is started any submissions from teams which are received after the contest clock reaches 00:00:00 are automatically marked as “deleted” (refer to the **Editing Runs** section, below). Since runs marked as deleted are not considered in scoring, late runs have no effect on the scoring of the contest.

Even though late runs are not considered in scoring computations, late runs are still captured by the system. This allows the Contest Administrator to make a decision later to accept such a run if desired; for example, if it was determined that due to site-specific conditions the run *should* have been accepted. Notifying the system to accept a late run can be accomplished simply by using **Edit Run** to change the run status to “not deleted”, along with setting the “Elapsed Time” (submission time) of the run as appropriate.

When PC<sup>2</sup> has been configured to automatically stop accepting runs at the end of the contest and a run is received after the contest ends, the team receives a notification back from the server informing them that the run was received too late to be counted.

Also, even though late runs are marked as deleted and hence not scored, they *are* automatically forwarded to the Judges. This allows the Judges to make a determination of what the result of the run *would have been* if it had been received within the time limit. However, even

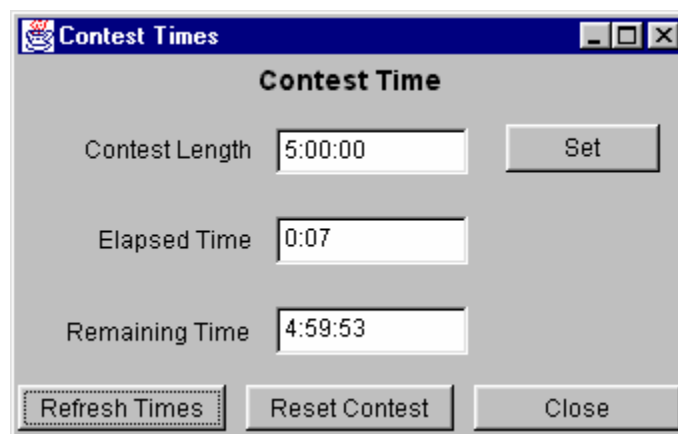
if a Judge responds “YES” to a run, it still will not be counted in the scoring if it was received after the clock reached zero and the “**Stop This Site Automatically**” box was checked.

If the “**Stop This Site Automatically**” box is *not* checked, then PC<sup>2</sup> continues to accept runs after the contest clock counts down past zero, and such runs are treated (and scored) exactly as runs submitted before the clock expired (except of course that their “submission time” will be larger). When the clock counts down past zero the Remaining Time is displayed as a negative number, in **RED**. Thus the absolute value of the clock display in this situation tells how long past the scheduled end of the contest the clock has continued to run.

Note that “contest time” in PC<sup>2</sup> is *site-based*, and also that the “automatic stop” feature applies only to the *current site*. These are intentional design features and are important in order to allow sites in a multi-site contest to make adjustments for site-specific conditions (such as the contest problems being handed out at different times at different sites, or the need to allow for a pause at one site). However, this means that contest clock coordination in a multi-site contest can be critical; see the section below on Multi-Site Clock Control if you are running a multi-site contest.

## 7.2. Contest Length

The **Set Contest Times** button on the Time/Reset screen displays the **Contest Time** dialog (shown below) which allows the Administrator to change the contest length to something other than the default length (5 hours).



Entering a new time in the form HH:MM:SS in the “Contest Length” field and then pressing the **Set** button will set the overall contest length to the specified amount of time. Setting a new contest length will also automatically update the Remaining Time field.

Normally the contest length is set only once, prior to the start of a contest, but this function can also be used to extend a contest after it has been started. However, the contest length can only be changed when the contest is *stopped*; the **Set** button will be disabled when the contest clock is running. Thus to extend a contest after it has started it is necessary to first pause (stop) the contest, then change the Contest Length, then start the contest again.

The time values in the **Contest Time** dialog do *not* update automatically when the contest is running; they display only the instantaneous time values at the moment the dialog is activated. If the contest is stopped when the dialog is activated, those times remain accurate indefinitely. If the **Contest Time** dialog is activated while the contest is running, the times can be updated by pushing the **Refresh Times** button. (Recall that the time values cannot in any case be changed unless the contest is stopped, so there is not usually any reason for needing to view this dialog while the contest is running.)

If a new Contest Length which is less than the current Elapsed Time is entered, the system displays a warning message to inform you that setting the contest length to a value less than the elapsed time will effectively mean the contest is over. If you decide to accept this condition, then if “**Stop This Site Automatically**” is checked then the contest will be automatically stopped. If automatic stop has not been enabled then the contest clock will indicate that time has moved past the end of the contest by displaying the Remaining Time as a negative number in **RED**.

The **Reset Contest** button is used when it is desired to effectively restart the contest clock from the initial conditions. It forces the Elapsed Time to zero and the Remaining Time to be equal to the Contest Length. **Reset Contest** is disabled when the contest clock is running.

### **7.3. Multi-Site Clock Control**

As described above, the contest clock in PC<sup>2</sup> operates on a per-site basis. That is, each site in a multi-site contest has its own contest clock, and PC<sup>2</sup> keeps track of “contest time” independently at each site. This is done to allow support for independent time-management constraints at different sites, and allows scoring to be done accurately without worrying about differences in timing between sites (e.g., a necessary pause at one site which does not affect other sites).

Each PC<sup>2</sup> site server determines the time of submission of a run from a team in terms of “contest elapsed time”, which means that a submission will be marked (“time-stamped”) according to the contest elapsed time at that site. The scoreboard in turn computes rankings based on this “submission time”, which means that overall (multi-site) rankings will be determined according to “contest elapsed time” at the site from which each run originated. This method puts teams at all sites on an equal competitive footing regardless of differences in the time at which the contest actually starts at each site.

However, this mechanism (keeping track of contest time independently at each site) can produce erroneous scoring results if the Contest Administrator does not take care to control the multi-site contest clocks correctly. Specifically, in a multi-site contest **it is important that the clock at each site be started at the moment the contest starts at that site.**

Typically, for example, a contest is deemed to have started at a site at the moment the contest problems are distributed to teams at that site. If this event (problem distribution; contest start) happens at different real times at different sites, then a Contest Administrator at each site should press the **Start Contest Time** button at that site precisely when the contest starts *at that site* – regardless of whether the contest has started simultaneously at other sites. In this way, runs

submitted by teams at each site will be correctly time-stamped with the true “contest elapsed time” as their “submission time”.

If all sites in the contest are fully connected, and the contest problems are handed out at the same instant at all sites, then a *single* Contest Administrator can easily coordinate the start of “contest time” at all sites correctly, simply by using the “**Start All Sites**” button on the **Time/Reset** screen. This button performs the same function as **Start Contest Time** except that it applies the corresponding action (starting the contest clock running) simultaneously to all connected sites. (Use the Admin **Sites** tab to determine which sites are connected; connected sites are those that have valid IP addresses displayed in the **Sites** screen).<sup>17</sup>

In a fully-connected multi-site contest where Contest Administrators at each site have agreed (e.g. by telephone or other method) on the precise instant at which all teams at all sites will receive the contest problems and thus the time at which the contest officially starts, having a single Contest Administrator press the **Start All Sites** button is the preferred (safest) way to coordinate the start of a contest. Likewise, if all sites are tightly coordinated, the **Stop All Sites** buttons can be used to stop the contest clock simultaneously at all connected sites.

However, if there are some sites which do not have network connectivity, or some sites at which the distribution of the contest problems (and hence the start of the contest) is delayed for some reason, it is *critical* that a Contest Administrator *at that site* makes *certain* that the contest clock is started *at that site* at the moment the problems are handed out (or whatever other criteria determine the moment in time when the contest starts at that site).<sup>18</sup>

We have seen more than one instance of a situation in a multi-site contest where the contest problems were handed out at all sites (hence, the contest is effectively under way at all sites), but one or more sites failed to notify PC<sup>2</sup> that the contest had started (either because the sites were not networked, or because the **Start All Sites** button was not used). In this case, if say 30 minutes elapsed before PC<sup>2</sup> is notified to start its contest clock at one site, then teams at that site will effectively get 30 “free” minutes – a run submitted 31 minutes after the problems were handed out will appear to PC<sup>2</sup> at that site to have been submitted “1 minute into the contest”.

### **Caveat Administrator:**

**It is critical that the PC<sup>2</sup> “contest clock” be started, at every site, at the time the contest starts at that site.**

In addition, recall that as noted previously the **Stop This Site Automatically** checkbox is entirely *local* in function. This box must be separately checked on the Admin client for each site where it is desired that the contest stop automatically when the Remaining Time clock counts down to zero.

---

<sup>17</sup> If a new site server is started after the **Start All Sites** button is pressed, it will be necessary to use an Admin client logged in to that new site to start the contest time for that site. Alternatively, you could use **Stop All Sites** then **Start All Sites** to start all sites including the new site; but this will cause all Clients/Teams to be unable to submit runs or clarification requests during the (brief) time between the Stop and Start.

<sup>18</sup> See the chapter on Loosely-Coupled Multi-site Contests for additional information on coordinating the contest clock in multi-site contests where sites which do not have direct network connectivity.

## 7.4. Practice Sessions: Resetting A Contest

In many contests, the overall contest activity starts with a “practice session” prior to the start of the actual contest. The primary objective of the practice session is to ensure that all teams are familiar with the operation of the contest environment (PC<sup>2</sup> in the present case) prior to the start of the real contest. The practice session might provide teams with a trivial “practice problem” to solve (“print your team name” or “read a file containing integers and print the sum of the integers”, for example), and then require all teams to login to PC<sup>2</sup> and test out the run submission mechanism by writing and submitting a solution to the practice problem. Some contests also require teams to practice using the PC<sup>2</sup> “clarification system” during the practice contest. A practice session also has the advantage of giving the Judges practice with how PC<sup>2</sup> works prior to the start of the real contest.

In order to run such a “practice contest” prior to the start of the real contest, it is necessary to configure PC<sup>2</sup> for the practice contest. Most of the configuration is identical to what is required for setting up the real contest – creating and configuring accounts, defining languages, etc. The only real difference is typically with the specification of the *problem set*: the practice problem must be configured into the system for the practice contest (it is undesirable to configure the real problems ahead of time, as this would mean the problem names would be visible to the teams during the practice). However, most configuration items other than the problem set are usually exactly the same during a practice contest as they are during the subsequent real contest.

At the end of such practice contest, it is necessary to “reset” the state of the system by removing from the database all runs, clarification requests, judgements, etc. which were submitted during the practice contest. However, it is at the same time desirable to *avoid* removing from the system the “configuration information” such as account names, passwords, language definitions, etc.

The **Time/Reset** screen contains a function to support the operations required to move from a practice contest to a real contest. Pressing the **Reset Site** button will clear the PC<sup>2</sup> database at that site of all runs, clarifications, and judgements from a practice contest. The **Reset Site** operation thus provides the ability to clear all data from the practice session and leave the system ready for entry of the real contest problem descriptions followed by a clean start of the real contest.

**Reset Site** affects only the current site (the site at which the Admin client is logged in). Because the operation it performs has such a substantial effect, the Contest Administrator is given a warning about the effects of the operation and is asked to confirm the intent. The Administrator should only proceed if it is clear that all activities related to the practice contest at the site are completed. Also, the **Reset Site** button can only be activated when the contest is in the “stopped” state at that site.

If the Contest Administrator confirms the **Reset Site** operation, the system creates a backup zip file containing the practice contest state, saves the zip file under the “archive” directory, and then clears the site database of all information specific to the practice contest (runs, clarification

requests, etc.). This leaves the system in a state where it is ready to have the problem descriptions for the real contest entered, and then the real contest can be started.<sup>19</sup>

The **Reset All Sites** button has a corresponding effect on all connected sites (that is, it notifies all other PC<sup>2</sup> site servers that they should also perform a **Reset Site** operation). If the contest is still running at another site when it receives such a “reset site” notification, *the contest is automatically stopped at that site*. The Contest Administrator is responsible for insuring that all practice contest operations *at all sites* are finished prior to invoking the **Reset All Sites** operation. The system provides one warning about the significant effect of this operation, and then if the operation is confirmed it instructs each site to perform a **Reset Site** operation (first stopping the contest at that site if it is still running).

Note that both the **Reset Site** and **Reset All Sites** buttons at a given site are disabled unless the contest is stopped at that site. However, as described above, when a **Reset All Sites** operation is selected, *it is not a requirement that the contest already be stopped at other sites*. A **Reset All Sites** operation will stop the contest at other sites automatically. Use this function with caution!

Note also that the **Reset Site** button is *not* the same thing as the “**pc2reset**” command (see Built-in Commands, earlier in this manual). Whereas the **Reset Site** button clears data related to a prior contest session, the **pc2reset** command *completely resets PC<sup>2</sup> to an initial installation configuration – it is the same as reinstalling the system from scratch*.<sup>20</sup> Specifically, for example, the **pc2reset** command deletes all account information, problem and language descriptions, etc. Be sure to use the proper command when attempting to “reset” your contest!

---

<sup>19</sup> Note: in the current version of PC<sup>2</sup> it is not possible to “delete” a problem description from the system without completely starting over. The operation of “replacing” practice contest problems with real contest problems essentially involves just changing the description (name) for each problem. This in turn means that, following a **Reset Site** operation, the “real contest” must have at least as many problems as there were in the “practice contest”. If this is not the case, simply enter blanks for the problem name of the extra problem descriptions).

<sup>20</sup> Except that system “.ini” files are not restored; any changes made to PC<sup>2</sup> “.ini” files will be retained even after executing a **pc2reset** command.



## 8. Monitoring Contest Status

### 8.1. Startup Status

The **Startup Status** tab on the main Administrator screen is used to track the status of Teams once a contest has been started. This is particularly useful during a “practice contest” held just prior to a real contest; it allows the Contest Administrator to verify that all teams have been able to login and use the basic PC<sup>2</sup> functions successfully.

A sample **Startup Status** screen is shown below. Initially all teams are displayed in RED, indicating that the team has not made any contact with the PC<sup>2</sup> server. When a team logs in, their display changes to YELLOW; when they have submitted at least one run or clarification request their display changes to MAGENTA or BLUE respectively; once a team has successfully submitted both a run and a clarification the display changes to GREEN, indicating the team has successfully performed all the basic PC<sup>2</sup> functions and should be ready to use the system in the real contest. (The screen below shows teams in each of these states, although it may be hard to read if you are not looking at a color copy of this manual.)

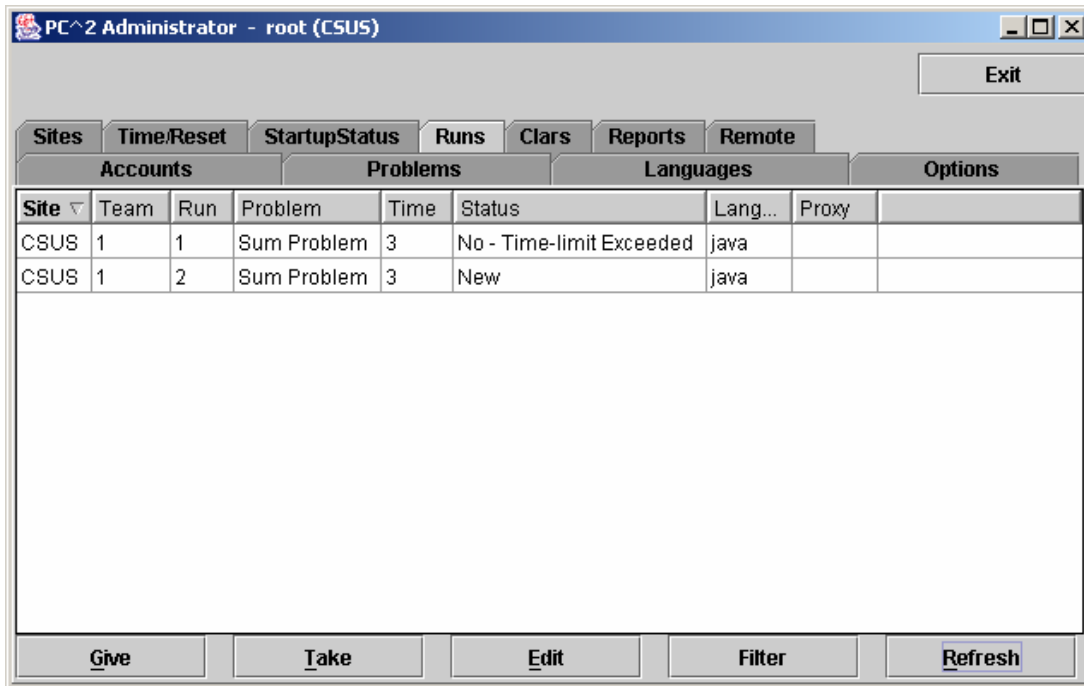


There is one situation which can arise in a contest whereby the **Startup Status** display will not accurately reflect the contest state. If a team logs in while there is no Admin client running, but does not submit any runs or clarifications, then logs out before the Admin client logs in, then that team login will not appear on the **Startup Status** display (the team will show as “No Contact”). A team must either already be logged in when the Admin client is started, or must login while the Admin is logged in, or must leave at least one run or clarification request in the system during the

team login, in order for the Admin client to recognize that the team has logged in. Note that under normal circumstances an Admin client is always started, and remains running, for the duration of a contest, so this situation will rarely occur.

## 8.2. The Runs Display

The **Runs** tab on the main Administrator screen displays a grid showing all the runs which have been submitted so far in the contest (from all teams, at all sites). The run display grid can be sorted on any column by clicking in the column header; clicking multiple times toggles the sort between ascending and descending order. The columns can also be resized by moving the column separators in the header. An example run grid is shown below.



The **Runs** display provides a number of capabilities for the Contest Administrator. One function of this display is to provide the ability to select a run which has already been judged (and hence no longer appears on the Judge's grid of available runs) and "give it back to the Judges" so that it may be re-judged. (Note that this assumes an Administrative decision to re-judge a run has been made for some reason; this is not a normal contest operation.)

To give a run back to the Judges, click on the row containing the run to select it, then click the **Give** button. This will cause the run to appear on the Judge's screens so that it can be selected and re-judged. (Note: when a Judge selects a run which has been sent for re-judging, a warning message is displayed on the Judge's screen asking for verification that the run really is intended for re-judging.) A run does not disappear from the Administrator's grid when it is sent for re-judging; the Administrator always has a complete listing of every run submitted in the contest, from all teams at all sites.

A second purpose of the **Runs** display is to allow the Administrator to "take a run away" from a Judge. This can be used, for example, to take back a run which was given in error to the

Judges for re-judging. Any time there is a run on the Judge's display grid which should not be there (because it has already been judged and is not intended to be re-judged, for example), click on the run in the Administrator's **Runs** display then click the **Take** button. This will remove the run from the Judge's screens.

### 8.3. Editing Runs

Another function of the **Runs** display is to allow the Contest Administrator to edit various parameters associated with a specific run – for example, to mark a specific run as “deleted” or to change the effective submission time of a run. This allows the Contest Administrator to make decisions regarding unanticipated situations affecting how a run should be considered in scoring.<sup>21</sup>

To edit a run, select the run in the **Runs** grid and then press the **Edit** button. This will bring up the following **Edit Run** dialog :

Run Information					
Site ID	1	Problem	Sum problem		
Team ID	1	Language	java		
Run ID	1	OS	Windows 2000		
Judgement	No - Time-limit Exceeded				
Elapsed Time	12	Proxy Site Number			
<input type="checkbox"/> Mark Run As Deleted					
<input checked="" type="checkbox"/> Suppress Team Notification					
Update		Execute		Cancel	

The “Problem”, “Language”, and “Judgment” drop-down lists allow the Administrator to alter the specification of the corresponding attributes associated with the run. It is allowable to change multiple attributes of a run during a single edit (although this would be unusual; normally, editing a run is done for a specific purpose such as correcting a judging error).

Changing the “Problem” attribute will have the effect of changing the way in which this run is considered in scoring: it will be counted as a run for the newly specified Problem (however, this will only have an actual effect on the scoring results if the Team has correctly solved the newly specified Problem; see the chapter on the Scoreboard for details). Changing the “Language” attribute will have the effect, if the run is subsequently re-executed, of changing the language definition (compiler invocation) used to compile and then execute the run. Changing the

<sup>21</sup> It is assumed that the Contest Administrator understands the ramifications of changing run attributes; for example, that changing the Elapsed Time affects the number of penalty points assigned to the run, changing the Problem affects how many runs a team has submitted for a given problem, etc. PC<sup>2</sup> is not smart enough to decide whether you *should* change the attributes of a run; it just gives you the *capability* to do so.

“Judgment” attribute will have the effect of causing the newly specified judgment to be the one used by the Scoreboard in determining whether the Team has correctly solved this problem.

“Elapsed Time” represents the elapsed time in minutes from the start of the contest (contest elapsed time, not counting minutes during which the contest clock was stopped) at which the run was received. The Administrator can change this value as desired. Note that Elapsed Time is considered the “team submission time” and is used to determine the calculation of penalty points assigned to the run.

“Proxy Site” is an internal value used to indicate whether, in a multi-site contest, this run has been handed off from its originating site to another site for handling; this should normally not be changed. (See the chapter on Loosely-Coupled Multi-Site Contests for further information on Proxy Sites.)

Checking the “Mark Run as Deleted” box will cause the run to be completely ignored in all scoring computations. Elapsed Time, Judgment value, and all other attributes of a run which is marked “Deleted” have no effect on scoring. (“Deleted” runs do not actually get removed from the database (nor from the **Runs** display), they are simply *marked* as such to indicate they should be ignored for scoring purposes.) If a run has been previously marked as deleted and subsequently the Mark Run as Deleted box is *unchecked*, the run is once again considered in further scoring computations, with no indication that it was previously “marked as deleted”.

The “Suppress Team Notification” checkbox is used to indicate whether or not the Team which submitted this run should be sent a notification of the “Judgment” value applied to this run after it has been edited. Normally, editing involves correcting an internal administrative error and it is not desirable to notify the Team of any editing, so the default operation is to suppress Team notification. If it is desired that the Team *should* receive a notification of the result of editing the run (for example, a judgment was changed from NO to YES and the Contest Administrator wishes the Team to know this), uncheck the “Suppress Team Notification” checkbox.

The **Execute** button on the **Edit Run** dialog allows the Contest Administrator to execute a run just as it would be executed on a Judge’s machine. This is useful, for example, when a Team submitted a run with an incorrect language specification (which most likely caused a Judge to render a “Compilation Error” judgment for the run), and it is desired to determine what the result would have been if the language specification had been correct. The Language drop-down list can be used to change the language attribute of the run, and it can then be executed at the Admin workstation. Note, however, that *the Execute function will only work correctly on the Admin workstation if the workstation has been configured with the necessary language compilers, in the same way as on a Judge’s machine.*

Once a run has been edited (and re-executed if desired), pressing the **Update** button will store the new specification of the run’s attributes in the database and, if team notification has *not* been suppressed it will send a notice of the run status to the Team. Note that once an update has been applied, the former state of the run is lost; there is no way to restore a run to a prior state once it has been edited (other than simply re-editing the run and changing the values back – but PC<sup>2</sup> does not keep track of the old run state once the **Update** button has been pushed).

## 8.4. Filtering Runs

It is frequently desirable to view a *subset* of the complete set of runs which are currently in the database. For example, the Contest Administrator may be interested in looking at all the runs for a given Problem, or all the runs from a given Team, or all the runs submitted during a specific window of time, or some combination of these.

The Administrator **Runs** display has associated with it a *filter* which can be used to apply a set of filtering criteria to the runs which are displayed.<sup>22</sup> Pushing the **Filter** button on the **Runs** display screen will produce the following dialog, which is used to set the filtering criteria:

Problems	Teams	Sites	Language	OS	Time
Bowling Penguins Sum of Squares	1:1 team1 1:2 team2 1:3 team3 1:4 team4 1:5 team5 1:6 team6 1:7 team7	CSUS Site 2 Site 3 Site 4	Java C++	Windows 2000	<input checked="" type="radio"/> All <input type="radio"/> Times From <input type="text"/> To <input type="text"/>
Select All Clear All	Select All Clear All	Select All Clear All	Select All Clear All	Select All Clear All	Reset
Select All			Clear All		
Update					Cancel

Selecting a set of items in the filter dialog indicates that those items *should* be displayed on the **Runs** grid. For example, to specify that the **Runs** grid should display all (and only) runs from Team 1 at site “CSUS” for the problem named “Penguins”, you would select “Penguins” in the Problems column, “Team1” in the Teams column, “CSUS” in the Sites column, and “All” in the Language, OS, and Time columns, and then press the **Update** button. The **Runs** grid would then apply the specified filter criteria to all runs, displaying only those that match the selected criteria. As another example, to display all (and only) runs which were submitted during the first 20 minutes of the contest, you would enter “0” in the “From” time field, “20” in the “To” time field, and select “All” in the Problems, Teams, Sites, Language, and OS fields, and then press the **Update** button.

The filter is designed with the expectation that at least one entry will be selected in each column. The “Clear All” button in each column is provided as a convenience to allow “deselection” of all entries in anticipation of then selecting one or more entries in the column.

<sup>22</sup> The Judge client provides a similar filtering operation; the discussion here on filters applies equally to the Judge.

However, clearing all criteria in a column does not make sense as a final filtering operation: if “Clear All” were allowed as the final choice in a column (that is, if no entry in the column was selected following a “Clear All” for the column), this would imply that *no* runs should be displayed (since no runs would match the filtering criteria of “nothing selected”). For this reason, the filter ignores a “Clear All” condition if it is the last operation performed on a column and instead effectively selects all the criteria in that column. In other words, pushing “Clear All” as the last operation on a column is effectively the same thing as pushing “Select All” on that column.

The same logic applies to the “Clear All” button near the bottom of the screen, which is used to initially clear all entries in *all* columns, in anticipation of then selecting at least one entry in each column. If this global “Clear All” is the last operation selected prior to pressing the **Update** button, the filter interprets this as a request to *disable the filtering operation entirely*; that is, to display all runs on the **Runs** grid. Thus, the global “Clear All” button is effectively a “Filter Off” (no filtering) button.

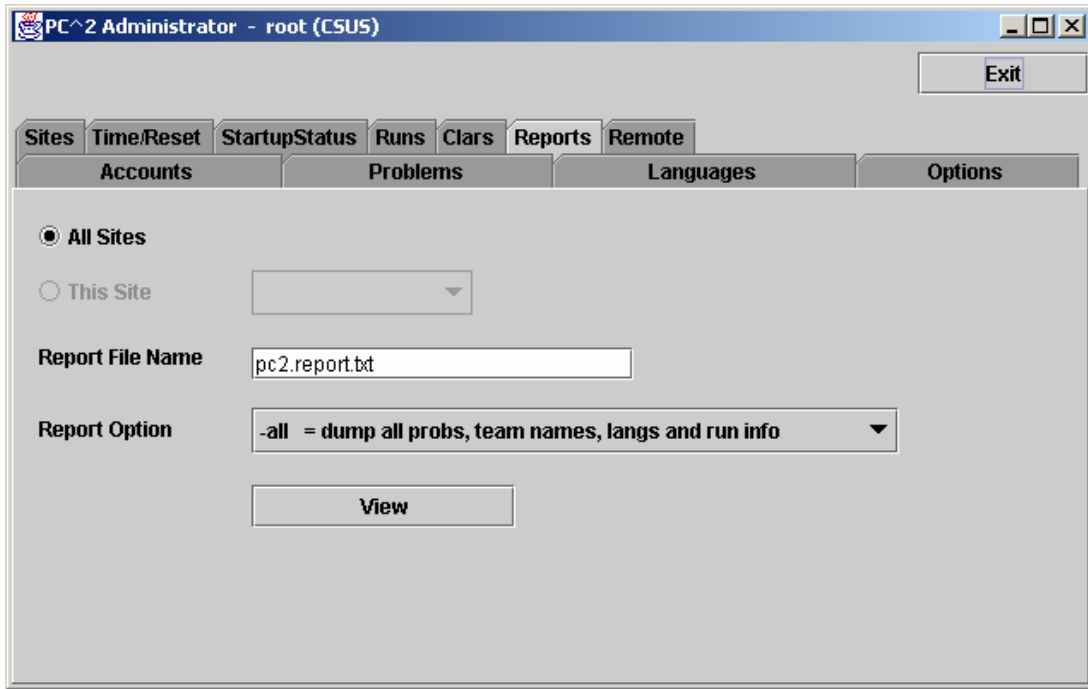
In order to remind the user when filtering of runs is taking place, any time a filtering operation has been selected the “**Filter**” button on the Admin screen will change color to blue and will indicate “on”, like so: **Filter (ON)**. Disabling filtering (using the “Clear All” button as described above) will return the **Filter** button to its normal state.

## **8.5. Clarifications**

The **Clars** tab on the main Administrator screen displays a grid showing all the Clarification Requests which have been submitted so far in the contest (from all teams, at all sites), in a format similar to the **Runs** grid. Like the **Runs** grid, the Clarification Request grid can be sorted and resized by manipulating the columns headers. The **Give** and **Take** function work like the **Give** and **Take** on the Runs pane. Note that there is no filter operation supported for Clarification Requests.

## **8.6. Reports**

The **Reports** tab on the main Administrator screen provides a variety of options for generating statistical reports about a contest, both during and after the contest. The **Reports** screen looks like the following:



The “**Report Option**” drop-down list allows choosing one of a number of different report formats. The list of available reports is summarized in the table shown below. Pressing the “**View**” button will pop up a display showing the content of the selected report, and will also write the selected report in text form to a file (located by default in the \$PC2HOME directory) using the file name given in the “**Report File Name**” text box.

Option	Description of Report
-all	Details of all problems, languages, teams, and runs
-complete	Same as -all but dumps all reports as well
-runs	Details of every submitted run
-clars	Details of all submitted clarifications
-prob	Details of problem definition info
-1	Summary of submissions by language
-3	Number of teams that solved each problem
-4	Number of correct and incorrect solutions by problem
-6	Summary submissions, solutions, times by team/problem
-5	Summary of who should have balloons (“Balloon Report”)
-7	Run listing per team by problem
-8	Run summary grid

## 9. The PC<sup>2</sup> Scoreboard

### 9.1. Overview

PC<sup>2</sup> contains a separate “Scoreboard” module which keeps track of the current standings in a contest. The scoreboard provides several functions: it automatically generates HTML pages describing the current state of the contest in a variety of formats; it generates email and/or printed “balloon notifications” (when that option has been selected by the Contest Administrator; see the section on “Options”, above); and it provides the capability to generate an “export” file containing contest standings data (in the form required for importing to the ICPC Registration system).

The HTML files generated by the scoreboard are placed in a directory named “html” (lower case) directly beneath the \$PC2HOME directory. Balloon notifications are sent to destinations as configured by the Contest Administrator. The export data file is placed in the \$PC2HOME directory.

The following sections describe the details of the generated HTML files and the export data file. They also describe the overall scoreboard operation, including the scoring algorithm used to compute contest standings.

### 9.2. Starting the Scoreboard

To start a scoreboard, go to a command prompt, be sure that the working directory is \$PC2HOME (the PC<sup>2</sup> installation directory) and type the command “**pc2board**”. This will start a PC<sup>2</sup> client expecting a scoreboard login.

Once the Client login window appears, enter the scoreboard account name “board1” and the board1 account password as defined when PC<sup>2</sup> accounts were created<sup>23,24</sup>. This will bring up the PC<sup>2</sup> Scoreboard display window (next page), indicating that the scoreboard program is running.

The scoreboard automatically generates a complete set of HTML files to the **html/** directory as soon as it is started. Thereafter it generates updated HTML files periodically according to an algorithm described below. Each time a new set of HTML files is generated the scoreboard display window is updated to show the most recent update time.

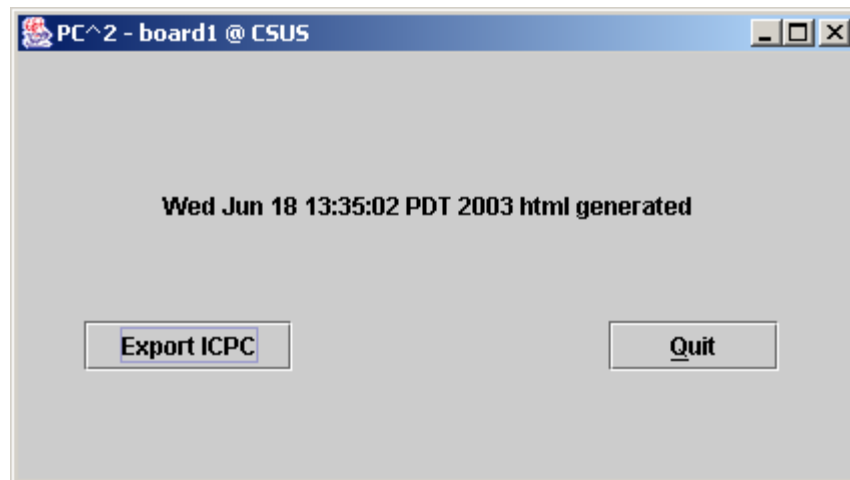
---

<sup>23</sup> Recall that by default, account passwords are the same as the account name. However, if the scoreboard account password(s) are not changed by the Contest Administrator, it would mean that any team could start a scoreboard running on their own machine, allowing them to look at the contest standings even during times when the Contest Administrator has decided to hide that information (such as near the end of the contest, which is the policy in some contests). We strongly recommend *changing* the scoreboard account passwords.

<sup>24</sup> The reason for using the “board1” scoreboard account is that “board1” is the only account which is capable of also generating balloon email/print notifications. If the contest is not using balloon operations, then all scoreboard accounts will work the same. However, we strongly recommend using only a *single* scoreboard machine, logged in as “board1”, for the entire scoring operation in a contest – even in the case of a multi-site contest.



Under normal circumstances it is only necessary to have a *single* PC<sup>2</sup> scoreboard running, even in a multi-site contest. The scoreboard automatically receives update information from every site server, and generates HTML files describing the overall contest status (including all sites). These HTML files can be copied to a publicly-accessible location for access by a browser (see below), so participants at any location can see the current standings. In addition, a single scoreboard (running under the “board1” account) can generate balloon notifications for all sites. Thus there is rarely a need for running more than one PC<sup>2</sup> scoreboard in a contest, and this is the recommended mode of operation.



### 9.3. Scoreboard Updates

Once it is running, the scoreboard waits passively until some contest event occurs which could alter the data it should display (this could be a submitted run, a judgement decision by the Judges, a change in a team name, or various other events). When any such event occurs, the scoreboard obtains from the server an update of the contest state, computes the new standings based on this information, and regenerates the HTML display files.

Because of the dynamic nature of scoreboard updates and the large number of events which can affect the scoreboard status, the scoreboard can generate a significant amount of communication traffic with the server. Further, in a multi-site contest, this traffic occurs with every site server before the scoreboard can regenerate the display files. In a slow or crowded network environment, or one subject to long delays, this communication overhead can be significant.

In order to reduce the amount of scoreboard traffic, the scoreboard contains a “throttling” mechanism to give the Contest Administrator control over the frequency of updates. This mechanism is based on a setting which can be specified in the **[client]** section of the `pc2v8.ini` file on the machine running the scoreboard module. If the **[client]** section defines an attribute of the form `minSecsBetweenUpdates=N`, then the integer value `N` is used as the minimum number of seconds which must elapse between scoreboard updates. If there is no definition of

`minSecsBetweenUpdates` in the `[client]` section of the `pc2v8.ini` file, the default value for `n` is 180 (three minutes).

The value of `n` is used to control the scoreboard in the following way. Every time an event occurs which could potentially alter the scoreboard state, the scoreboard checks to see if it has been at least `n` seconds since the last time it updated the display. If `n` seconds have not elapsed since the last update, the event is ignored for the moment. If `n` seconds HAVE elapsed, then the scoreboard collects from all servers the current state of the contest and creates a new set of HTML files based on that data.

Thus, the default condition in PC<sup>2</sup> is that the scoreboard is updated at most every three minutes, but the Contest Administrator can arrange that this time is increased or decreased as desired. Note that setting the value of `minSecsBetweenUpdates` to *zero* will have the effect of eliminating throttling; the scoreboard will compute new standings every time any relevant event occurs (but at the expense of generating a potentially significant amount of network traffic).

There are two exceptions to the rule of limiting scoreboard update frequency. First, any time a “YES” judgement is issued by the Judges, the scoreboard *unconditionally* updates, regardless of the throttling value. Second, any event which occurs while the contest is in a STOPPED state also will force an unconditional scoreboard update (this includes the condition that the contest is stopped because it has ended). In all other circumstances the scoreboard updates in a time-constrained manner as described above.

## 9.4. Scoreboard HTML Files

Each HTML file generated by the PC<sup>2</sup> scoreboard is a complete stand-alone HTML document (i.e. is bracketed by `<html> ... </html>` tags). Each document `<head>` includes a `<title>` tag, into which PC<sup>2</sup> places the Contest Title as specified by the Contest Administrator (see **Options**, above). Each document `<body>` contains an imbedded `<table>` holding contest status information.

The `<table>` in each different HTML file contains a different set of contest information, such as team rankings, run submission statistics, etc. (see below). The information outside the `<table>` can be edited/replaced as desired, for example by adding additional header information, frames, or any other HTML constructs. However, it is important to keep in mind that the set of HTML files is *completely regenerated* on every scoreboard update; changes made manually to an HTML output file will only persist until the next scoreboard update.

The following HTML files are always generated by the scoreboard:

File Name	Table Contents
<code>full.html</code>	Columns showing rank, team display name, number of problems solved, and penalty points, with rows ordered by rank. This is the standard “contest standings” display.

File Name	Table Contents
<b>fullnums.html</b>	Same as <b>full.html</b> except that the Team Display Name is preceded by the Team Number followed by a dash.
<b>final.html</b>	Standings as they are displayed at the conclusion of the ACM ICPC World Finals – Penalty Points are considered and displayed only for the top ten teams; the remaining teams who have solved at least the median number of problems are grouped alphabetically by number solved; the rest of the teams are listed as “Honorable Mention”.
<b>sumtime.html</b>	A grid showing, for each team and each problem, the number of runs submitted by the team for the problem, and, if the team has solved the problem, giving the contest elapsed time of the team’s solution.
<b>sumatt.html</b>	A table similar to the <b>sumtime</b> table but instead of giving the time of solution for solved problems simply indicates “Y” or “N” according to whether the team has solved the problem.
<b>summary.html</b>	A table combining the <b>full</b> and <b>sumtime</b> displays described above – that is, a showing rank, team display name, number of problems solved, penalty points, and number of runs submitted and solution time for each problem, with rows ordered by rank.

The most common way to take advantage of the HTML files generated by the PC<sup>2</sup> scoreboard is to run a separate external process (e.g. a batch script) which repeatedly copies the current HTML files to some external location, reformats them if desired, and makes them available to teams and spectators using a browser.

There are several advantages to this method of operation. First, the details of the appearance of the scoreboard can be customized by the Contest Administrator external to PC<sup>2</sup>. Thus the Contest Administrator can choose to take advantage of the full set of scoreboard screens, or can choose to omit some or all of them. In addition, it is not necessary for teams, judges, spectators, etc. to run separate PC<sup>2</sup> scoreboards, since most users will already have access to a browser. The Contest Administrator can arrange that the external scoreboard script builds the desired scoreboard display and puts the resulting HTML in a standard public location accessible to all user’s browsers.

## 9.5. Scoring Groups

In addition to the above files, the scoreboard can generate files showing rankings based on the concept of “groups” or “regions” with which a team is associated. For example, in the ICPC World Finals, teams compete not only for placement in the overall world-wide standings, but also

among teams from their own region of the world for the regional championship.<sup>25</sup> Other examples include situations where it is desirable to break contest teams up into separate groups based on level of experience (e.g. “lower division” and “upper division” students), and in multi-site contests where it is desirable to be able to display rankings that show only those teams participating at a given site.

Every PC<sup>2</sup> account has associated with it a “Region ID” identifying the “region” or “group” to which the account belongs. By default all accounts have a Region ID of zero (which is considered a valid Region ID); hence all teams are by default in the same “region” (group). Changing the Region ID for a particular team associates that team with other all teams having the same Region ID (see the section on Editing Accounts, earlier in this manual).

By default, the scoreboard ignores Region IDs. However, it is possible to make the scoreboard pay attention to Region IDs (that is, pay attention to team groupings), and when this is done it will cause the scoreboard to automatically generate additional HTML files, as follows:

File Name	Table Contents
<b>regions.html</b>	A single table showing the top team in each region (team grouping). <sup>26</sup>
<b>regionX.html</b>	This represents several different HTML files, one for each region “x” ( <b>region1.html</b> , <b>region2.html</b> ,.... ). Each <b>regionX.html</b> file contains the same contents as the file <b>full.html</b> described above, except that the data is restricted to teams which are defined as belonging to “region x”. <sup>27</sup>

There are two ways to cause the scoreboard to pay attention to regional groupings of teams (and hence generate the additional region HTML files). One is to use the “import ICPC data” function on the main Administrator screen. The imported ICPC data contains information identifying for PC<sup>2</sup> both the number of regions into which teams have been partitioned, and the “region ID” and “region name” associated with each team. This information both enables the automatic generation of the regional HTML files, and tells the system how determine the region with which a given team is associated.

---

<sup>25</sup> Currently the ICPC defines six world “regions” (called “Super Regions” for historical reasons), and awards the Super Regional Championship to the top team in each Super Region as part of the Word Finals competition each year. The current ICPC Super Regions are: Africa and the Middle East, Asia, Europe, Latin America [comprising Central and South America], North America, and the South Pacific.

<sup>26</sup> Note: if multiple teams are tied for the top position in a region after taking into account number of problems solved, penalty points, and time of last correct submission, the region entry in the **regions.html** file will simply display <TIE> without displaying a team. If this happens, check the **regionX.html** file to determine the standings.

<sup>27</sup> In the current version of PC<sup>2</sup> this description is taken literally: the data in each **regionX.html** file is the same as that which appears in the **full.html** file. In particular this includes ranking numbers; thus, **regionX.html** files display a team’s ranking in the overall contest, not just in the region as might be expected. However, teams will be ordered in descending rank order so it is easy to discern relative rankings.

The second method which can be used to enable regional HTML file generation is to use attribute assignments in the **[board]** section of the scoreboard's `pc2v8.ini` file. Attribute values can be defined for specifying the number regions, the name of each region, and the ID number associated with each defined region.

If generation of region/grouping information is enabled via `pc2v8.ini` attributes (as opposed to being enabled via an Import ICPC data operation), the Contest Administrator must also take care to assign a Region ID to each account according to the region to which the account belongs. All accounts in the contest (even accounts across multiple sites) which belong to the same region or group should have the same Region ID number, since this is how the scoreboard determines the association between an account and its region. (This assignment is handled automatically when the Import ICPC data operation is used.)

For example, if it was desired to separate all contestants into “Lower Division” and “Upper Division” groups, all accounts used by teams in the “Lower Division” might be given Region ID values of “1” while all accounts used by teams in the “Upper Division” might be given Region ID values of “2”. Note that in a multi-site contest, all Lower Division teams would need to have the same Region ID (2), regardless of the site at which they were located.

As another example, if it was desired to be able to display standings in a multi-site contest on a per-site basis, the accounts for all teams at (for example) Site 1 should be given Region ID values of “1” while the accounts for all teams at Site 2 should be given Region ID values of “2”, etc. See the appendix on **pc2v8.ini attributes** for further information.

## 9.6. Scoring Algorithm

The algorithm used in PC<sup>2</sup> to compute Rank and “Score” (Penalty Points) is the one used in the ACM ICPC World Finals, which is as follows:

- 1) Teams are ranked according to the number of problems solved; a team solving more problems is always ranked higher than a team solving fewer problems.
- 2) Within a group of teams solving the same number of problems, teams are ranked by increasing “Penalty Points” (that is, the team with the lowest number of Penalty Points is ranked highest within the group). Teams only accrue Penalty Points for problems which the team has **solved**; unsolved problems do not affect the scoring in any way. Teams accrue Penalty Points for solved problems in two ways:
  - One point for each minute elapsed from the start of the contest until the problem was solved (the time of SUBMISSION is counted as the “time solved”; it does not matter how long it took the Judges to judge it).
  - A specific number of penalty points for each INCORRECT submission submitted to the Judges *prior to a correct solution* for the problem (runs submitted after a correct solution are not counted in the scoring).<sup>28</sup>

---

<sup>28</sup> In the ICPC World Finals the number of penalty points for each incorrect submission prior to solving a problem is always 20; this is also the default value in PC<sup>2</sup> (although PC<sup>2</sup> allows the Contest Administrator to change this value using a Configuration Option).

- 3) If two or more teams have the same number of solved problems and exactly the same number of Penalty Points, ties are broken in favor of the team with the earliest time of the last correct submission (that being the time when the team “finished” the contest).

## **9.7. Export Data File**

The scoreboard provides the ability to generate a text file containing data describing the current contest standings. Since this capability was provided for purposes of exporting contest results back to the ICPC Registration system, the button on the scoreboard display which is used to invoke it is labeled “**Export ICPC**”. However, the use of the export operation is not restricted to interfacing with the ICPC Registration system; it simply generates a file containing text records made up of comma-separated fields.

The export data file contains, for each team, a record giving the number of problems solved, the total number of penalty points accrued on solved problems, and the time of last submission of a correct solution (used in the ICPC World Finals as a tiebreaker). See the Appendix on Import/Export Interfaces for further details on the format of the export data file.

To generate the export data file, simply press the **Export ICPC** button on the scoreboard display. This causes the scoreboard to generate a file named “**pc2export.dat**” containing export data. The file is located in the \$PC2HOME directory.

## 10. Loosely – Coupled Multi-Site Contests

### 10.1. Overview

PC<sup>2</sup> will support multi-site contests running in either “tightly-coupled” or “loosely-coupled” mode. Tightly-coupled mode means that there is an active network connection between sites, whereas loosely-coupled mode means that there are separate sites running the same contest but with no direct network connectivity between the sites. In either mode, each site is always running its own PC<sup>2</sup> Server.

In tightly-coupled mode, all PC<sup>2</sup> events which occur at any site are automatically transmitted by the system to all connected sites. For example, runs submitted by a team at one site automatically appear on the screens of judges at all sites; changes in the scoreboard state at one site are automatically displayed at other sites; etc. A tightly-coupled contest is established by setting up one primary server together with one or more secondary servers configured with `pc2v8.ini` entries which cause the secondary servers to connect with the primary server at startup, as described earlier in this manual.

If a single contest<sup>29</sup> is being run at multiple sites which do not have direct connectivity with each other, it is possible to manage the overall contest by using PC<sup>2</sup> running in *loosely-coupled* mode at each site. In loosely-coupled mode the servers do not directly communicate with each other. Instead, each separate site is carefully configured so that it is *aware* of the other sites in the contest but does not directly communicate with them. To replace the functionality which is lost due to servers not communicating directly with each other, PC<sup>2</sup> provides an off-line external mechanism for merging data from each site into a single set of overall contest standings. This means that every site can see the overall contest standings even though there is no direct network connectivity between the sites.

In order to set up a loosely-coupled multi-site contest, the server at each site must be made aware of the other sites, without being told to attempt a network connection to the other sites.<sup>30</sup> This is done by including in each server’s `sitelist.ini` file a listing of ALL servers (sites) in the contest, while at the same time NOT including a `remoteServer=xxx` entry in *any* server’s `pc2v8.ini` file. Thus *every* server is operated as a “primary” server in a loosely-coupled contest.

Once a contest is set up to operate this way it is possible to arrange for “off-line” communication and sharing of standings data between the sites – including the ability to generate a “merged scoreboard” showing the combined results of competition taking place at multiple loosely-coupled contest sites. The following sections describe the details and constraints involved in doing this.

---

<sup>29</sup> Meaning a contest using the same problem set and being run at approximately the same time at all sites.

<sup>30</sup> In principle it is possible to run a contest in both tightly- and loosely-coupled mode; that is, where some sites are tightly-coupled with each other and there are also some loosely-coupled (non-connected) sites in the contest. In this case, all the tightly-coupled sites are considered as a single “pool” of sites, and the “pool” is essentially the same as a single ‘site’ which is loosely-coupled with other sites (or pools).

## 10.2. Master Site

To support loosely-coupled multi-site operation, it is useful to have a separate site in the contest acting as the “Master Site”. The Master Site is responsible for off-line acquisition of contest data from each of the other sites, for merging the data into a “single contest” view, and for generating the “combined standings scoreboard”. PC<sup>2</sup> has built-in functions to perform these operations automatically.

While it is theoretically possible to have one of the active competition sites in the contest also act as the Master Site, we recommend instead that a completely separate PC<sup>2</sup> “site” be set up strictly for purposes of acting as the Master Site. From the point of view of other sites, this Master Site is just another site in the contest; it should be listed in the `sitelist.ini` file of all sites, just as all other sites must be listed. The Master Site can be physically located at one of the actual contest sites; however it is generally less confusing – and less error prone – if the Master Site is logically distinct from an active competition site.

PC<sup>2</sup> must be installed and configured at the Master Site just like it is at all the sites where teams are competing. The procedure for installation and configuration of PC<sup>2</sup> at the Master Site is identical to that of any other site. Once PC<sup>2</sup> is installed, start a PC<sup>2</sup> server at the Master Site, then start an Admin client. Login using the Admin **root** account and configure the contest at the Master Site in the normal way. The Master Site must have the same PROBLEM configuration as the other contest sites<sup>31</sup>, but it will not normally have any languages defined (since those may differ between sites), nor will it normally have any Team or Judge accounts (it should have Scoreboard and Admin accounts).

## 10.3. Loosely – Coupled Startup

When a contest is run in *tightly*-coupled mode, an Administrator at one site can effectively “start” the contest at all sites simultaneously – by pressing the “Start All Sites” button on the Administrator “Time” screen. This causes PC<sup>2</sup> to automatically start the “contest clock” on all servers, thus synchronizing all sites. However, in *loosely*-coupled mode, there is no way for the clock to be started simultaneously in PC<sup>2</sup> since the servers by definition cannot communicate with each other in loosely-coupled mode. *This can create a very bad situation if the humans involved do not take care to avoid it.*

Specifically, if the contest data from various loosely-coupled sites is to be transported to other sites using the Export/Import and Merge functions (for example, in order to produce an “overall scoreboard”; see below), it is *critical* that all sites start their PC<sup>2</sup> contest clock *at the time the contest starts at that site* – meaning, some human at each site must push the “Start Contest” button at the moment the contest problems are handed out and the contest begins *at that site*. Otherwise, *even if all teams receive the contest problems at precisely the same moment in real time and have precisely the same amount of time to work on them*, the merged scoreboard results will be inaccurate due to incorrectly synchronized starts at various sites!

---

<sup>31</sup> In fact, it is *critical* in a loosely-coupled contest that *all sites* enter the contest problems (see the **Problems** tab on the Administrator main screen) in the *same order*. That is, “Problem #1” must be the same problem at all sites.



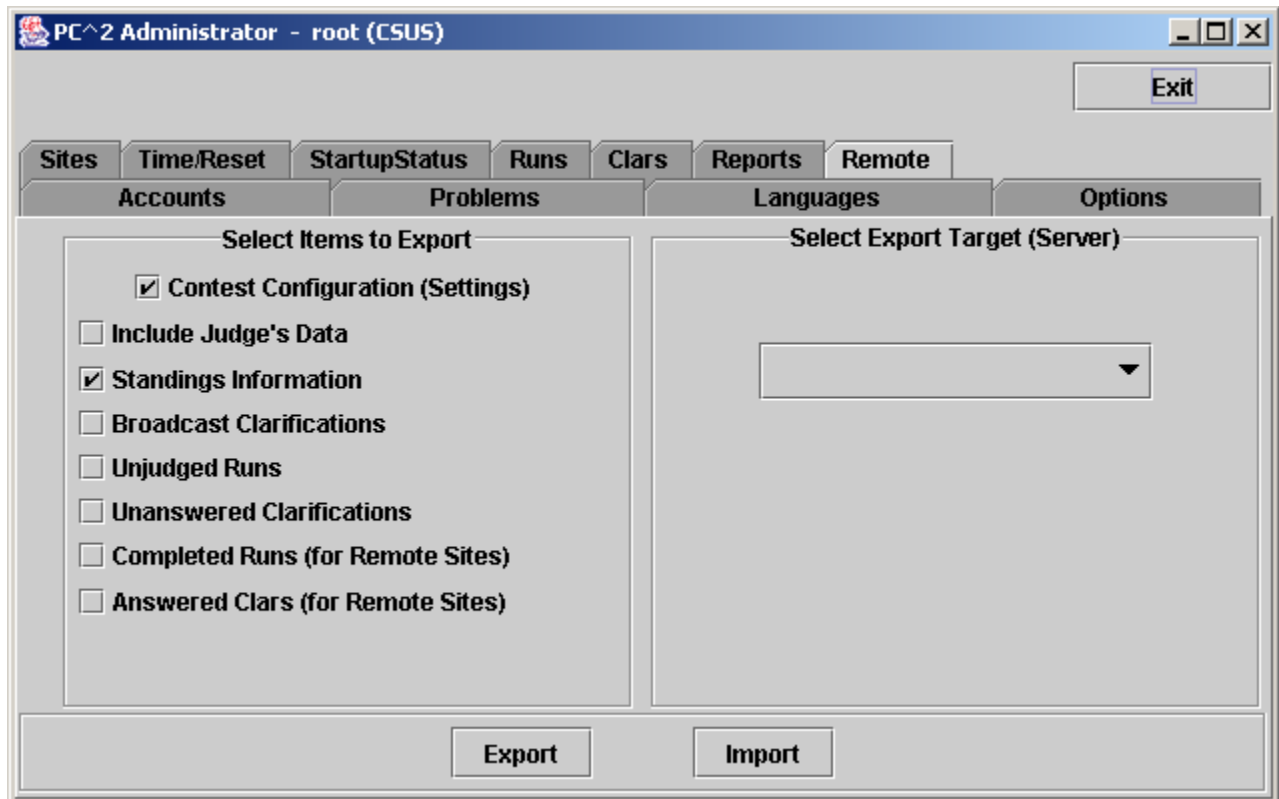
For example, if all sites hand out the problems at the same instant but one site doesn't start the PC<sup>2</sup> clock for another 30 minutes, then every run submitted at that site will have a timestamp which is off (by 30 minutes) from the other sites. Since scores are computed based on times (and hence timestamps), the merged scores will be *wrong*. We have seen this happen – more than once – at loosely-coupled contests, and there is no way to fix it except to edit all the run timestamps manually through the Administrator interface at the Master Site.

Therefore we offer the following **WARNING**: The Contest Administrator needs to make *sure* there is someone at *each* loosely-coupled site that is responsible for pushing the PC<sup>2</sup> “Start Contest Time” button at the moment the contest starts at that site. Please see the additional discussion of this topic under “Starting the Contest”, above. You've been warned....

## 10.4. Generating Merged Standings

The most common reason for using PC<sup>2</sup> in loosely-coupled mode is to provide the ability to generate a single scoreboard showing the merged standings of all sites in the contest, even though the site servers do not have active connections to each other. In order to generate such a merged scoreboard, each site must “export” its current configuration and standings to the “Master Site”, which “imports” the data from each site and generates a single scoreboard from the merged data. This export/import/merge operation can be done on the fly, at any time (and multiple times) during a contest.

To do the export, a Contest Administrator at each site must select the “**Remote**” tab on the main Administrator screen, which produces the screen shown below. The Contest Administrator should insure the “**Contest Configuration**” and “**Standings**” items under “Items to Export” are *both*



selected (checked), and then click the **Export** button. This will pop up a dialog prompting for a file name into which the export data should be written. (Note: do not confuse this Export file, generated for communication between loosely-coupled sites in a multi-site contest, with the “ICPC Export Data” file described in the chapter on the Scoreboard and in the Appendices; they are not related to each other.)

The “Select Export Target” list on the right side of the Remote screen is not used when generating an export file for merging standings.

Once the export data file is created it must be transferred by external means (ftp, email, etc.) to the Master Site. The mechanism by which this is accomplished is up to the Contest Administrator; it is outside the scope of PC<sup>2</sup>.

Once the Master Site has received an exported file from a site, the file is imported into the Master Site by clicking the **Import** button and selecting the file which was received from the remote site. The data in the file will be merged with the current standings in the Master Site.

*NOTE: it is important that each site in a loosely-coupled multi-site contest be properly (identically) configured in terms of the number and order of the contest problems. When an **Import** is done at the Master Site, the import includes an update to the contest configuration as defined by the imported data; if the data between various sites is inconsistent then the merged results will be similarly inconsistent.*

To generate a merged scoreboard (that is, a single scoreboard reflecting the combined standings of teams competing at all “loosely-coupled” sites), start a PC<sup>2</sup> Scoreboard on the Master Site and login to a Scoreboard account. This will generate standard PC<sup>2</sup> HTML display files as described in the chapter on the PC<sup>2</sup> Scoreboard. The data in the scoreboard HTML files will reflect the combined standings of all sites whose data has been imported into the Master Site. The HTML files can then be published via a web server, from which they can be accessed by teams at all sites, or they can be transferred back to each individual site by external means (ftp, email, etc.) and displayed locally by the site.

## **10.5. Cross-Site Judging**

Many loosely-coupled contests operate by having judges available at every site to handle submitted runs and clarification requests from teams at that site. However, this is not always the case. Suppose that in a loosely-coupled contest there are not judges available at all sites, or that for purposes of consistency all judging is to be done by contest staff located at a single one of the sites. Under these conditions, teams at a site where there are no judges will submit their programs (to their PC<sup>2</sup> site server), but there will be no judge connected to the server to accept the runs.

In addition to supporting merged-scoreboard operations as described in the previous section, the Remote Export/Import mechanism can be used to support cross-site judging in such situations. The Contest Administrator at the team’s site can export the submitted runs from that site to a site where there *are* judges. (Typically this would be the “Master Site”, but it need not be). Exporting team runs is done by selecting “**Unjudged Runs**” as one of the “Items To Export” prior to pressing the “**Export**” button. Similarly, Clarification Requests which have been

submitted by teams (but have not been answered because there are no judges at that site) can be exported to a site where there are judges, by selecting “**Unanswered Clarifications**” prior to generating the export file.

In order for PC<sup>2</sup> to be able to keep track of runs and clarifications which have been “exported” for handling at another site, it is necessary that the export of runs and clarifications be made “destination site-specific”. That is, when exporting runs and/or clarifications it is necessary to identify precisely the site to which the runs/clarifications are being exported. This is done by using the “**Select Export Target (Server)**” drop-down list to choose the site to which the export data is going to be sent.

The receiving (target) site in this situation is known in PC<sup>2</sup> terminology as the “proxy site” for the run/clarification, and the exported runs are said to be “proxied” to the specified target site. The generated export data file will be rejected during an import attempt by any site other than the one for which it was targeted. Also, it is not allowed to “proxy” (export) a run or clarification to more than one site at the same time. While this “targeted export” mechanism forces somewhat of a constraint on cross-site operations, in that the Contest Administrator must decide on a policy governing where exported runs/clarifications are to be sent, the export/import facility still provides a substantial contest management facility in an environment where there is neither active network connectivity nor judging capability at all sites.

Once a set of exported runs/clarifications has been imported into the destination site, the runs and clarifications will immediately be displayed on the screens of all judges connected to that site and can be handled by the judge just as if they had been generated locally (i.e. by a team connected to that site). Once the judge finishes with the run or clarification, it goes back to the local server just like any other submission. At this point the Contest Administrator at the proxy site would use the Export function, selecting “**Completed Runs**” and/or “**Answered Clars**” for export, and specifying the originating site as the target.

Once the exported file is shipped back to the originating site and imported, the results (run judgements and/or clarification responses) will be processed at the originating site just as if the sites had been actively connected.<sup>32</sup> In this way the Export/Import facility provides a sort of “virtual connection” between otherwise disconnected sites in a loosely-coupled contest.

The ability to export/import runs can also be used to support a “verification” operation by a “Head Judge” at the Master Site. Under this scenario, the Contest Administrator would export from each site (and import into the Master Site) the completed runs for each site. The Head Judge can then re-execute selected runs as desired to verify correctness of the final standings.

Note however that in order to perform such a “verification” operation, the Head Judge would need to be running on a machine with a configuration which is identical to that of the team which submitted the run at the remote site. If different sites have different environments – some Windows and some Unix, for example – the Head Judge will need to have access to both platforms in order to execute runs in both environments.

---

<sup>32</sup> There is one exception to this rule: Teams do not receive an “instant message” notification of completed items (run judgements and clarifications) when the items are imported from a remote site. However, the item status (result) does display as usual on the corresponding Team grid if the Team presses their “refresh” button.

## Appendix A – pc2v8.ini Attributes

As described in the chapter on PC<sup>2</sup> Initialization Files, the **pc2v8.ini** file consists of up to six sections: **[server]**, **[client]**, **[admin]**, **[judge]**, **[team]**, and **[board]**, with each section containing one or more "attribute assignment" statements of the form **attributeName=value**. Lines in the file which begin with a “#” or “;” character are ignored, as are blank lines. Attribute names (left side of the equal-sign) are not case sensitive; however, string data on the right side of the equal-sign *is* case sensitive.

The following list gives the attributes which can be defined in each section of the **pc2v8.ini** file, along with a description of their function. Some attributes may appear in more than one section.

### [server] section attributes:

**debugLevel=<integer>**

Sets the level of debugging output sent to the log file (**pc2serv.log**). Higher integer values cause more detailed output to be written to the log. The default value if this attribute is not specified is 2. Values above 99 are not useful (produce no additional output). Values in the range 10-20 produce useful information without generating reams of internal data; a good value for most contests is 13. If you are having problems, raising this attribute value may help identify the problem. This attribute can also appear in the **[client]** section.

**consoleLevel=<integer>**

Sets the level of logging output sent to the server console screen. Higher integer values cause more detailed output to be written to the screen. The default value if this attribute is not specified is 2. Values above 99 are not useful (produce no additional output). Values in the range 5-20 produce useful information without generating reams of internal data; a good value for most contests is 5. If you are having problems, raising this attribute value may help identify the problem. This attribute can also appear in the **[client]** section.

**site=<site\_name\_string>**

Tells the server the name of the site which it is serving. The value **<site\_name\_string>** should be replaced with the exact character string identifying the Site (i.e., one of the lines from the **sitelist.ini** file). Site names must be unique across all contest sites. Note that site name strings are *case-sensitive*. This attribute can also appear in the **[client]** section.

**port=<portNumber>**

Tells the server the port number on which it should expect to be contacted by clients, and by other PC<sup>2</sup> servers in a multi-site contest. This attribute may be omitted from the **pc2v8.ini** file, in which case it defaults to 50002. Note that if you choose to assign a specific port number, then all clients and other servers contacting this server must also be told to use this same port number (this is specified with the “**server=**” attribute in the case of clients, and with the “**remoteServer=**” attribute in the case of other servers). Note also that if you choose to assign a specific port number, the port number should be greater than 49151 according to the conventions established by the IANA (Internet Assigned Numbers Authority)<sup>33</sup>.

**remoteServer=<IPAddress>:<portNumber>**

Tells the server the IP address and port number of a remote PC<sup>2</sup> server at another site which it should contact in order to join a multi-site contest. The **<portNumber>** must be specified and must match the port number being used by the server at the remote site. The appearance of this attribute makes this server a “secondary” server; if this attribute is not defined in the **[server]** section then this server is a “primary” server and waits passively to be contacted by other site servers.

### **[client] section attributes:**

**site=<site\_name\_string>**

Tells the client the name of the site to which it belongs. The value **<site\_name\_string>** should be replaced with the exact character string identifying the Site (i.e., one of the lines from the **sitelist.ini** file on the server). Site names must be unique across all contest sites. Note that site name strings are *case-sensitive*. This attribute can also appear in the **[server]** section.

**server=<IPAddress>:<portNumber>**

Specifies the IP address and port number at which the client module should attempt to contact the PC<sup>2</sup> server. Every client module **MUST** have a “**server=<IPAddress>:<portNumber>**” entry in the **[client]** section of its **pc2v8.ini** file. The IP address and port number must correspond to the address of the machine running the PC<sup>2</sup> server and the port number at which the server on that machine is expecting to be contacted.

---

<sup>33</sup> <http://www.iana.org/assignments/port-numbers>

**workDir=<fullPathName>**

Specifies the “working directory” for the client. This directory is used as the default directory in dialogs which require the user to select a file. If this attribute is not specified, the default is the directory from which the client was started. This attribute can also appear in the **[team]** section.

**minSecsBetweenUpdates=<integer>**

Specifies the number of seconds which the scoreboard should wait before performing an update when an event occurs. Events which occur in less than the specified amount of time since the previous event are ignored, except for certain key events such as the occurrence of a “YES” judgement which *always* force a scoreboard update. The default value for this attribute if it is not specified is 180 (three minutes). Setting this attribute to have the value zero effectively forces the scoreboard to update on the occurrence of *any* event. (See the chapter on the scoreboard for information on the ramifications of setting this attribute.)

**minSecsBetweenUpdates=<integer>**

Specifies the number of seconds which the scoreboard should wait before performing an update when an event occurs. Events which occur in less than the specified amount of time since the previous event are ignored, except for certain key events such as the occurrence of a “YES” judgement which *always* force a scoreboard update. The default value for this attribute if it is not specified is 180 (three minutes). Setting this attribute to have the value zero effectively forces the scoreboard to update on the occurrence of *any* event. (See the chapter on the scoreboard for information on the ramifications of setting this attribute.)

### **[board] section attributes:**

**regionIdCount=<integer>**

Specifies the number of distinct groupings or “regions” into which contest teams have been assigned. If this value is greater than zero it causes the scoreboard to generate “region” scoreboard HTML display files for each defined “region” (group) of teams. The default value for this attribute is *zero*, which suppresses the generation of region files by the scoreboard. If this attribute is defined then the **regionNameCount** attribute (below) must also be defined. See the section on the scoreboard for further details.

**regionNameCount=<integer>**

Specifies the number of distinct region names which are defined. This attribute must be defined if **regionIdCount** (above) is defined, and it must have the same value as **regionIdCount**.

**regionID $\underline{X}$ =<integer>**

Specifies the ID number associated with region "X". Note that this represents the occurrence of *multiple* attribute assignments, one for each defined region. In other words, there must be exactly as many **regionID $\underline{X}$**  attribute assignment statements in the **pc2v8.ini** file as the value of the **<integer>** which appears in the **regionIdCount** statement (see above). The integer which appears on the right side of a **regionID $\underline{X}$**  statement can be any non-negative value; different regions are not required to have adjacent integer values (only unique ones). Note, however, that the values of "X" on the left side in the set of **regionID $\underline{X}$**  attributes should start at 1 and have increasing consecutive integer values (see the example below).

**regionName $\underline{X}$ =<string>**

Specifies the name associated with region "X". Note that this paragraph represents *multiple* attribute assignments, one for each defined region. In other words, there must be exactly as many **regionName $\underline{X}$**  attribute assignment statements in the **pc2v8.ini** file as the value of the **<integer>** which appears in the **regionNameCount** statement (see above). Different regions should have unique name strings.

To clarify the use of region groupings, the following example is provided. This set of **pc2v8.ini** entries will cause the scoreboard to generate regional HTML files for two regions: one consisting of all teams whose Region ID is three (grouped into a region named "Upper Division"), the other consisting of all teams whose Region ID is seven (grouped into a region named "Lower Division") :

```
regionIDCount=2
regionNameCount=2
regionID1=3
regionName1=Upper Division
regionID2=7
regionName2=Lower Division
```

### [admin] section attributes:

**useTeamName**=<*anything*>  
**useSchoolName**=<*anything*>  
**useSchoolShortName**=<*anything*>  
**useTeamAndSchoolName**=<*anything*>

These four attributes are used in conjunction with importing ICPC Registration data into PC<sup>2</sup>. If any one of these attributes is defined (that is, if the attribute name on the left side has been assigned some non-empty string on the right side), then the Admin client uses the corresponding item from imported ICPC data to determine the Display Name which will be used for a team on the scoreboard. For example, if “**useSchoolName=true**” was specified (and none of the other attributes were specified), then the team’s full school name would be used on displays. The default value if none of these attributes is specified is **useTeamName**. If more than one of these attributes is specified then the highest one in the list as given above has precedence. These attributes are only useful if ICPC import data has been used to set the various choices of names available for display. See the section on importing ICPC information for further details.

### [judge] section attributes:

**defaultAnswer**=<*string*>

Specifies the string which will be sent to a Team (from a Judge) in response to a Clarification Request if the Judge pushes the “Default Answer” button rather than typing in a specific answer to the Clarification Request. The default value for this attribute if it is not defined (and hence also the default value for the Judge’s “Default Answer” button) is “No Response”. Note that this attribute must be defined in the **pc2v8.ini** file for *every judge who will be responding to clarifications*.

### [team] section attributes:

**workDir**=<*fullPathName*>

Specifies the “working directory” for the team. This directory is used as the default directory in dialogs which require the team to select a file. If this attribute is not specified, the default is the directory from which the Team client was started. This attribute can also appear in the **[client]** section.



## Appendix B – Networking Constraints

As mentioned in the beginning of this manual, PC<sup>2</sup> modules must be able to communicate with each other via TCP/IP: clients must be able to communicate with their servers, and servers in a multi-site contest must be able to communicate with other servers. If client machines reside on the same LAN segment as their server, and if all servers have publicly routable IP addresses which can be reached by other servers, then communication should work with no problems.

However, given the wide variety of network configurations which can exist – firewalls, NAT, and VPNs, just to name a few – there may be some constraints in a given network setup which will cause problems in setting up a contest using PC<sup>2</sup>. In order to understand how to avoid (or circumvent) these problems, it is useful to have some understanding of how PC<sup>2</sup> networking is implemented.

PC<sup>2</sup> is written in Java and uses Java RMI (Remote Method Invocation) for communication between modules. RMI is a high-level remote-communication protocol which encapsulates information regarding connections between separate network nodes (machines) and allows a Java program running in a Java Virtual Machine (JVM) on one node to make direct calls to methods in a program running in a separate JVM, typically on a separate network node<sup>34</sup>.

When a PC<sup>2</sup> server is started, it “registers” itself with a process running on the same machine, known as the Java RMI Registry (or just “registry” for short). If there is no registry running on the machine when the server starts, then the server arranges to start the registry.

Registering the server with the registry includes notifying the registry of the port number at which remote PC<sup>2</sup> modules (clients or other servers) expect to make initial contact. This initial contact port number is obtained by the server from the **pc2v8.ini** file. Registration also includes obtaining back from the registry a port number at which the server should listen for remote connections. This is the port number that will be given to any remote modules which contact the registry asking for a connection to the server. **This port number is generated automatically (internally) by the Java RMI Registry; PC<sup>2</sup> has no control over the port number.**

Subsequently, when a remote PC<sup>2</sup> module (a client or another server) wishes to contact the PC<sup>2</sup> server, it first contacts the RMI Registry on the server’s machine. The registry in turn gives the client (or other server) a handle to the initial server. This handle contains the IP address and port number for the server; the remote module can then directly communicate with the server using the IP address and port number provided by the RMI Registry. Note again that this mechanism is part of the Java RMI protocol; it is outside the scope of PC<sup>2</sup> and PC<sup>2</sup> has no control over it.

There are several important ramifications of the RMI protocol implementation in terms of effects on PC<sup>2</sup> networking. First, it means that PC<sup>2</sup> will not work if there is a firewall between the server and the remote module attempting to contact it and the firewall is blocking access to the

---

<sup>34</sup> While the remote method is always by definition contained in another JVM, it does not necessarily have to be on a separate network node; it could be in another JVM running on the same machine. This is why it is possible to have a PC<sup>2</sup> server and multiple PC<sup>2</sup> clients running on the same machine; they run in separate JVMs and communicate with each other via RMI.

port(s) which the RMI Registry has assigned for communication between the server and the remote module.

Note that this does *not* refer to the port number which the Contest Administrator specifies in the “**port=**”, “**server=IP:port**”, and “**remoteServer=IP:port**” entries in the **pc2v8.ini** file. Those port numbers are effectively the port at which the server notifies the RMI Registry to listen for connection attempts. The RMI Registry essentially “hands off” any such connection attempts to a (random) port, notifying both the server and the remote module of that new port number. Since PC<sup>2</sup> has no control over the generation of that new port number, it means that **the firewall must allow connections between any port number which could be generated by the RMI Registry.**

Another constraint on networking has to do with NAT (Network Address Translation). If one PC<sup>2</sup> module is operating from behind a NAT box, then when the RMI Registry issues a handle to a remote module giving the IP address of the first module, that IP address will be the “untranslated” address – i.e. the “local address” behind the NAT box. When the handle is eventually received at a remote destination on the other side of the NAT box, the local address will not have been translated by NAT (since the address is encapsulated inside an RMI packet). This means that the remote destination will have no idea of the correct “return address” (the public address to which addresses behind the NAT box are translated) to use for communication. In other words, it is a limitation of RMI protocol that it does not work through NAT. Thus it is not possible to run PC<sup>2</sup> modules when the communication needs to go through NAT to succeed.

While RMI protocol will not work through NAT, it *will* work across a VPN (Virtual Private Network). This is a potential solution to both the firewall and NAT problems. If the Contest Administrator arranges for a VPN connection between PC<sup>2</sup> modules, those modules will be able to communicate using RMI just as if they were on a public network. An explanation of how to set this up is beyond the scope of this manual, but we have tried it and it works; a good network administrator should be able to set up such a VPN.

## Appendix C – Restarting A Server

As described in the section on **Server Startup**, when a PC<sup>2</sup> server is started the user will be prompted with a question, as follows:

**Are there already servers up (Y, N, or Q) ?**

If this is a case of *restarting* a server which was *previously running* in a multi-site contest but crashed or was shut down for some reason, and if there are other servers still up and running in the contest, then this is said to be a case of a server attempting to *rejoin* a contest. Special steps are necessary in order to allow a server to rejoin a contest.

First, since modules in the current version of PC<sup>2</sup> do not have the capability of detecting a loss of network connectivity<sup>35</sup>, the servers which are still running will think the server for the site being restarted is still “connected” (even though it has been shut down). The running servers will therefore refuse to accept a connection from the rejoining server. Thus, before the server being restarted can rejoin the contest, the other servers must be told to drop the (bogus) connection to that server. This operation is referred to as a “forced logout” of the disconnected server.

To perform a forced logout on the server being restarted, it is necessary to use a PC<sup>2</sup> “Admin client” connected to some *active* server. To do this, start a PC<sup>2</sup> “Admin client” connected to an active site, by setting the “**site=**” and “**server=**” entries in the [**client**] section of the **pc2v8.ini** file for the Admin client so that they point to the active server at another site. Next, go to the **Sites** tab on the main Administrator screen, and if the server is listed as “active”, select the site and click on the **Logoff** button.<sup>36</sup> It is only necessary to perform the forced logoff operation on one active server; that server will automatically notify the other servers to drop the dead server connection.

Once the active servers have dropped the dead connection to the server being restarted, that server can rejoin the contest. To do this, first answer “Y” to the question shown above. At this point one of two conditions holds for the restarting server:

1. The server’s **pc2v8.ini** file indicates it is a secondary server, by virtue of containing an active “**remoteServer=**” entry. In this case the server will automatically attempt to contact the primary server at the IP:port address given in the **remoteServer** entry in the **pc2v8.ini** file. Assuming the primary server is up, the restarting server will automatically rejoin the contest.
2. The server’s **pc2v8.ini** file indicates that it is a primary server, because it does *not* contain any **remoteServer** entry. In this case the server will issue a second prompt:

### **Enter Server to Contact:**

In this case the user should enter the IP address and port number, in the form “**IPAddr:port**” (without the quotes), of some other server which is currently up and running in the contest. The reason this is necessary is that otherwise the restarting

---

<sup>35</sup> Yea, yea... we already said *we’re working on it!* (Please keep in mind that this whole thing is a *volunteer* project...)

<sup>36</sup> Alternatively, contact a person located at the other site and ask them to perform the “forced logoff” operation on the dead server connection.

server, since it is ‘primary’ according to its `pc2v8.ini` file, would simply sit and wait to be contacted by other servers – which would require shutting down and restarting the (secondary) servers still running at other site(s).

Note: an alternative way to accomplish the same thing (that is, to allow a primary server to rejoin a running contest) would be to edit the server’s `pc2v8.ini` file prior to the restart and add a new `remoteServer=IP:port` entry, thus effectively switching this server from primary to secondary status. This would effectively cause the entire contest to be running nothing but “secondary” servers. This would not be a problem as long as no complete restart is needed; as mentioned previously, once the system is up and running in “steady state”, all servers act as peer equals. If a complete restart became necessary, it would require at least one of the site servers to be re-designated as “primary” so that it could be started and wait for other (secondary) servers to contact it.

Rejoining a contest can cause confusion if the Contest Administrator who is restarting the server forgets to insure a “forced logout” of the dead connection. In this case, the active remote server will refuse to accept the connection from the restarted server. This in turn causes the restarting server to log a message describing what has happened, and then simply give up (terminate). If you are attempting to restart a server and rejoin a contest and it seems to simply quit for no reason, check the contents of the file `pc2serv.log`. If the log ends with a series of messages like

**Error Contacting/logging-in to other server  
Server from [this site] already logged in??  
Server halted, have someone logoff this site.**

then the server to which you are trying to connect thinks you are already connected and you need to perform a “forced logout” of your server connection on the running server(s).

## Appendix D – PC<sup>2</sup> Server Command Line

The PC<sup>2</sup> server is a Java class named “Server” in a subpackage named “server” of the package “pc2”. The command to start a server is

```
java pc2.server.Server
```

typed in the \$PC2HOME directory. This is in essence the command invoked by the PC<sup>2</sup> “built-in” command `pc2server`.

The server accepts a number of command line options when it is started. One option is `-h` (or `--help`), which produces the following “usage” output:

```
$ java pc2.server.Server --help
CSUS Programming Contest Control System
mailto:pc2@ecs.csus.edu
Version 8.4 20030501 01 (May 1st, 2003 3:12pm) Java ver 1.3.0_02

Usage: Server [-v|--version|-h|-help] [-first|-join] [-site sitename]

-h   - this message (--help)
-v   - print version information (--version)

-first - no prompt, this server is a first server (no join)
-join  - no prompt, this server contacts a remote server
-site sitename - add to contest with site named sitename
```

As seen from the “usage” output, the server also accepts the following command line options:

- v (or --version) : prints the version identification for the current PC<sup>2</sup> server
- first : indicates that this server is a primary server and should not attempt to contact any other servers (ignores any `remoteServer=` attribute in `pc2v8.ini`)
- join : indicates that this server is a secondary server and should attempt to contact a remote server to join a contest. Uses the `remoteServer=` attribute in `pc2v8.ini`, if there is one, to determine what remote server to contact; if no attribute is defined, prompts at the console for IP:port connection information
- site <sitename> : sets the sitename for the site being served by this server (overrides any specification of site name in `pc2v8.ini`)

## **Appendix E – ICPC Import/Export Interfaces**

### **E.1 Importing ICPC Registration Data**

As mentioned earlier in this manual, PC<sup>2</sup> was designed for supporting the ACM International Collegiate Programming Contest, including its local and Regional contests worldwide. The ICPC maintains an online Contest Registration system which is used by Regional Contest Directors (RCDs) around the world to manage participation in the various ICPC Regional Contests. PC<sup>2</sup> provides interfaces to import contest registration data from the ICPC Registration system, and also to export contest results back to the ICPC web site.

To import ICPC Registration data to PC<sup>2</sup>, the RCD must first log into the ICPC Registration system and download the “PC<sup>2</sup> Initialization” zip file which is automatically created and updated as registration data changes occur.<sup>37</sup> Once the PC<sup>2</sup> Initialization data file is downloaded, it should be “unzipped” at any convenient location.

Unzipping the PC<sup>2</sup> Initialization data file will produce three separate files: “PC2\_Contest.tab”, containing details about the organization of the contest (such as the formal name of the contest); “PC2\_Site.tab”, containing data identifying the sites in the contest; and “PC2\_Team.tab”, containing data about the teams that are registered in the contest.

The PC2\_Team.tab file consists of text-based tab-delimited records, one record for each team registered in the contest. The tab-delimited field contents of each record are as follows:

- 1) An integer giving the ICPC Team ID (note that this is *not* the same thing as the PC<sup>2</sup> Team ID; see below).
- 2) An integer giving the ICPC Region ID (the “region” or “group” to which the team belongs).
- 3) A single character indicating the Team’s “registration status” for the contest – typically either ‘P’ (pending), ‘A’ (accepted), or ‘C’ (cancelled). PC<sup>2</sup> ignores records containing ‘C’ in this field.
- 4) A string giving the Team Name; for example, “The Top Coders”.
- 5) A string giving the full name of the Team’s school; for example, “California State University, Sacramento”.
- 6) A string giving the Team’s school name in “short form”; for example, “CSUS”.
- 7) A string giving the Team’s school’s URL; for example, <http://www.csus.edu>.
- 8) A string giving the Team’s school’s country code (three letters); for example, “USA”.
- 9) A single character ‘Y’ or ‘N’ indicating whether the Team’s school has a graduate program. PC<sup>2</sup> ignores this field (but it must be present).

---

<sup>37</sup> The PC<sup>2</sup> initialization data is contained in a file whose name on the ICPC web site is typically something like “CI532.zip” – “Contest Information for contest number 532”. However, both the naming convention for the file, and the exact location of the file on the web site, are outside the scope of (i.e., not controlled by) PC<sup>2</sup>.

PC<sup>2</sup> can “import” this team data and use it to define things such as the team’s Display Name on the scoreboard. PC<sup>2</sup> will associate consecutive records in the **PC2\_Team.tab** file with consecutive PC<sup>2</sup> accounts – the first record in the file will automatically be associated with “team1” at the site specified in the record, etc. (Note that PC<sup>2</sup> does *not* use the ICPC Team ID field for purposes of identifying a team.)

If the contest administrator wishes to associate registered teams with PC<sup>2</sup> accounts in a different order, the **PC2\_Team.tab** file can be edited (rearranged) to present the records in the desired order – the team which should be “team1” in PC<sup>2</sup> should appear first in the file, etc. Alternatively, if the **PC2\_Team.tab** file is copied to a new file named “**\_PC2\_Team.tab**”, the new file can be edited by adding a new leftmost column containing a PC<sup>2</sup> team number. PC<sup>2</sup> checks for the existence of this file and, if it is present, uses that file not only to load the ICPC data but to determine, based on the value in the first field in each record, to which PC<sup>2</sup> account each record in the data applies.

Once the PC<sup>2</sup> Initialization data has been unzipped and the **PC2\_Team.tab** file edited if desired, the next step is to press the “Import ICPC Data” button on the PC<sup>2</sup> Administrator main screen (under “Manage Accounts” on the **Accounts** tab pane). This will cause PC<sup>2</sup> to display a “file selection” dialog; this dialog is used to identify for PC<sup>2</sup> the set of .tab files which should be imported. To make this identification, simply navigate to the directory containing the unzipped ICPC PC<sup>2</sup> Initialization data files, and select any one of the “.tab” files in that directory. This will cause PC<sup>2</sup> to load all three of the initialization files. Note: all initialization “.tab” files must all reside in the *same directory*.

If this is the first time an ICPC Import has been done, PC<sup>2</sup> will show a dialog displaying the names of the contest sites as defined in the ICPC data and asking the administrator to provide a specification of the association of those site names with the site names which PC<sup>2</sup> already knows about (from its **sitelist.ini** file). Once this is complete, PC<sup>2</sup> saves the region-association information<sup>38</sup> and then finishes the association of ICPC Registration data with PC<sup>2</sup> accounts; the results can be viewed on the **Accounts** tab on the Administrator main screen, just as if the data had been entered manually. (Note: the account display on the Administrator screen is not dynamically updated upon completion of an ICPC import; click to another view and then back to the “Manage Accounts” view in order to see the imported data.)

Note that each record in the ICPC PC<sup>2</sup> Initialization data contains *multiple* names associated with each team – the team name, the full name of the school, and a short version of the school name. PC<sup>2</sup> can be configured to use any one of these names, or a combination of them, as the name to be displayed on the scoreboard. See the appendix on **pc2v8.ini** file entries, under the **[admin]** section, for details.

PC<sup>2</sup> expects quotation marks in the team data to be “quoted”. That is, if any field in the data contains a quotation mark (”), then (1) the entire field must be surrounded by an additional matching set of quotation marks, and also (2) each quotation mark which is part of the data must

---

<sup>38</sup> The region-site association data is saved in a file named “**\_pc2\_site.tab**”. If this file exists when an Import ICPC Data operation is performed, PC<sup>2</sup> does not prompt for the region-site association information, but uses the contents of the “**\_pc2\_site.tab**” file instead. To force it to reread new imported site information (for example on a subsequent Import), delete the file “**\_pc2\_site.tab**”.

be doubled. Thus for example a team name like **The "TOPS" Team** should appear in the import data file as **"The ""TOPS"" Team"**. If the PC<sup>2</sup> Initialization file exported from the ICPC web site contains data with quotes, it may be necessary edit the data by hand (or load it into a program such as Microsoft's Excel and then save it) to insure that quotation marks are properly formed. If this is not done prior to importing the data into PC<sup>2</sup>, you may see "format error" messages in the log file, and the data containing the quotes will not be displayed properly.

## **E.2 Exporting Contest Data**

Pressing the "**Export ICPC**" button on the scoreboard display causes the scoreboard to generate a text file containing the current contest standings in a form required by the ICPC World Finals results display algorithm.<sup>39</sup> This file is made up of a series of text records, one record for each team in the contest. Each record contains a set of comma-separated fields. The fields in each record are:

- 1) ICPC Team ID (note: *not* the PC<sup>2</sup> team number).
- 2) Rank in contest (this field is blank if the team falls in the "Honorable Mention" category as defined by the scoring display algorithm for the ICPC World Finals; see the description of the file **final.html** in the chapter on the PC<sup>2</sup> Scoreboard for details).
- 3) A non-negative integer giving the number of problems the team has solved.
- 4) A real number giving the total number of penalty points accrued by the team.
- 5) A real number giving the time of the last submission by the team, taking into account only the problems which the team has solved (used as the ICPC World Finals tiebreaker determination).

The records in the file are sorted by team rank based on problems solved and penalty points. In particular, teams that fall into the "Honorable Mention" category as defined by the ICPC World Finals results display algorithm will still appear in rank order in the file, as defined by number of problems solved, penalty points accrued, and tie-breaking time of last submission.

The exported data file contains all the information necessary to update the ICPC web site registration system with the results of a contest. It is primarily intended to provide an automated mechanism for Regional Contest Directors to post the results of Regional Contests.<sup>40</sup> The export data file can also be imported into any program that wishes to make use of the standings data.

Note that if no "**ICPC Import**" operation was performed prior to invoking the "**Export ICPC**" operation, then PC<sup>2</sup> will have no record of the ICPC Team ID associated with each team. In this case the "Team ID" value in field #1 in the exported file will be zero. This problem can be circumvented in contests other than Regional Contests (that is, contests where there is no ICPC data to import) by creating a local version of the "**PC2\_Team.tab**" file, entering the appropriate **PC<sup>2</sup>** team account number in lieu of the ICPC Team ID. This will cause the "**Export ICPC**"

---

<sup>39</sup> Actually, the export data file is automatically generated (updated) each time the contest standings change; pressing the "Export ICPC" button simply displays a message showing its location.

<sup>40</sup> However, to our knowledge as of this writing (July 2003) the upload functionality on the ICPC web site has not yet been implemented. This is a separate operation outside the scope of PC<sup>2</sup>; RCDs should check with the ICPC Contest Manager as to the availability of this function.



operation to generate a file containing all the data necessary to compute complete contest standings utilizing PC<sup>2</sup> account numbers.

## Appendix F – Validators

### F.1 Overview

PC<sup>2</sup> allows the Contest Administrator to configure each problem so that it has associated with it a *validator* program whose purpose is to help automate the judging process. A “validator” is a program which is given, as input, the output of the execution of a run (that is, the output of a program submitted by a team).

Validators can operate in one of two ways: passively or actively. A passive validator is a program which accepts the team program’s output and displays it in some useful form for examination by the human judge. An active validator is a program which not only accepts the team program’s output but contains logic designed to make a determination, according to some set of rules, regarding the correctness of the output. An active validator can also return the result of its determination to PC<sup>2</sup>.

An example of a passive validator would be the use of a “side-by-side” file comparison utility which displays the team program output beside an “answer file” provided by the Judges ahead of time. This allows the judge to use the validator to examine the output and compare it with the correct answer, perhaps taking advantage of special capabilities of the validator program (for example, syntax-driven color highlighting of differences between the team program’s output and the correct answer file). When the judge is satisfied, the validator is terminated and the judge uses the PC<sup>2</sup> Judge client interface to enter a judgement.

An active validator is one which automatically examines the team program output and makes some determination of its correctness. The validator must contain program logic which directs how it determines correctness. This logic could be hard-coded within the validator (in which case the validator is almost always problem-specific), or could be more general (for example, instructions directing application of “difference testing” between the program output and a Judge’s “answer file”). In either case an active validator is one which is designed to make an “automated” determination of the correctness of the output of a team’s program.

For active validators, a set of conventions defined within PC<sup>2</sup> provide a mechanism for the validator to return to PC<sup>2</sup> an indication of what judgement it thinks should be applied to the run. However, such a validator response is never blindly accepted as the final determination of the judgement to be applied to a run. Rather, the validator output is displayed to the human judge as a “recommendation”. It is ultimately up to the human judge to accept the validator recommendation (or not). That is, it still requires the judge, at a minimum, to press a button to accept the validator recommendation.

### F.2 Validator Configuration

By default there is no validator attached to (associated with) a contest problem in PC<sup>2</sup>. Validators can be attached to a problem by the Contest Administrator by using the **Validator** tab on the **Edit Problem** dialog. This displays the Validator configuration screen, as shown below. When the Contest Administrator configures a problem to use a validator, then when a Judge executes a

team program the specified validator will automatically be invoked as soon as the team program completes execution.

The Validator screen allows the Contest Administrator to choose one of two options for attaching a validator to the problem being defined: either the simple built-in PC<sup>2</sup> validator, or an arbitrary separate external program. If a separate external program is selected it can be either a standard system program (such as a “diff” utility or a split-screen file-compare utility) acting typically as a passive validator, or it can be a program written specifically for the validation of output from programs for this particular contest problem, i.e. an active validator.

The Contest Administrator can choose whether or not to display the validator result to the judge. Checking the box “Show Validation To Judges (SVTJ)?” when configuring a validator will cause the response returned by the validator to appear on the Judge’s display when the run finishes executing. If the checkbox is unchecked, the validator result will not be visible to the judge. The checkbox should normally be unchecked when using passive validators, which do not return meaningful result information (if SVTJ is checked and a passive validator is used, the response displayed to the judge will always be “Undetermined”).

The following sections describe the built-in PC<sup>2</sup> validator, the use of an existing external program as a validator, and the creation and use of a special-purpose problem-specific validator. For problem-specific validators, the ICPC standard for interfacing such validators to a contest control system such as PC<sup>2</sup> is described.

The screenshot shows the 'Edit Problem' dialog box with the 'Validator' tab selected. The dialog has two main sections: 'PC<sup>2</sup> Validator' and 'External Validator'. In the 'PC<sup>2</sup> Validator' section, the 'Do not use a validator' radio button is selected. Below it, the 'Use PC<sup>2</sup> Validator' radio button is unselected. The 'PC<sup>2</sup> Validator' section contains a 'Validator Options' dropdown menu set to 'None selected' and an unchecked 'Ignore Case In Output' checkbox. The 'External Validator' section is unselected and contains a 'Validator Program' text box with a browse button (...), a 'Validator Command Line' text box containing the placeholder text '{:validator} {:infile} {:outfile} {:ansfile} {:resfile}', and a checked 'Show Validation To Judges (SVTJ)' checkbox. At the bottom of the dialog are 'Update' and 'Cancel' buttons.

### **F.3 PC<sup>2</sup> Built-In Validator**

PC<sup>2</sup> contains a built-in validator which can be used to compare the output of a submitted program with the “answer file” provided by the judges (specified when a contest problem is configured). The built-in validator is essentially a *very simple* version of the standard Unix “diff” file comparison utility. Selecting the “**Use PC<sup>2</sup> Validator**” option enables a drop-down list showing the various comparison options, which are listed in the following table.

<b>PC<sup>2</sup> Validator Option</b>	<b>Meaning</b>
1	Perform a standard “diff”: a character by character comparison of files differences
2	Ignore whitespace at start of file
3	Ignore leading whitespace on lines
4	Ignore all whitespace on lines
5	Ignore empty lines
6	2 then 3
7	2 then 4 (implies 3)
8	2 then 5
9	3 then 5
10	4 then 5 (implies 2 then 3)

When using the PC<sup>2</sup> built-in validator it is important to understand the exact meaning of the options which specify combinations (options 6 and higher). In particular, “X then Y” means that the validator first checks the output using case X and *then* checks the output using case Y, independently. Note that this is *not* the same thing as performing a single test applying both conditions simultaneously. This can produce unexpected results.

For example, “check the output, ignoring white space at start of file” followed by “check the output, ignoring empty lines” is not the same as “check the output ignoring white space at start of file and also ignoring empty lines”; if there are empty lines in the output file but not in the answer file, then when it validates using the criteria “ignore white space at start of file” the output will FAIL due to the empty lines in the answer file. ***Be sure you understand the operation of the built-in validator before using it!***<sup>41</sup>

The PC<sup>2</sup> built-in validator is an active validator – meaning that it automatically returns to the Judge a recommendation for the judgement which should be applied to the run whose output was diff’d against the answer file. If SVTJ was checked when the validator was attached to the problem, then the returned recommendation is displayed for the judge.

---

<sup>41</sup> Yes, we know it’s a crummy implementation. More motivation for you to write your own validator, as described in the following sections.

## **F.4 Using An Existing External Tool As A Validator**

If the Contest Administrator selects **Use External Validator** on the Validator configuration screen, it tells PC<sup>2</sup> to use an external program as a validator for the problem being configured. In this case PC<sup>2</sup> needs to be given two sets of information regarding the external program which is to be used: (1) the name of the file containing the validator, and (2) the command sequence which PC<sup>2</sup> should use when invoking the validator.

To understand how this works, suppose it is desired to use an external program named “*gvim*” as a validator for a certain contest problem. Further, suppose *gvim* is a program which can be given an argument, “-d”, along with two additional arguments specifying file names, and that what it does in this case is display the contents of those two files side by side, highlighting the differences.<sup>42</sup>

What the Contest Administrator wishes to do is have PC<sup>2</sup> invoke *gvim* every time a run for this particular problem is submitted and executed by a judge, and pass to *gvim* two files: (a) the output of the team’s program, and (b) the answer file (correct output) for the problem (provided ahead of time by the judges). This will allow a judge to compare the team’s output with the correct output, using the capabilities of the *gvim* program to assist them.

Since our hypothetical *gvim* program already exists and is not (likely) written to understand how to make programming contest-related decisions about the correctness (or not) of the differences between two files, we will use *gvim* as a passive validator (one which does not return a value to PC<sup>2</sup>). In that case the Contest Administrator would *uncheck* the SVTJ box, and then would select **Use External Validator** on the Validator screen.

In the **Validator Program** textbox the Contest Administrator enters the file containing the *gvim* program. This might be something like

```
/usr/local/bin/gvim
```

or

```
C:\Program Files\gvim\gvim.exe
```

Use the “browse” button (the button showing “...”) to select the validator program file name. Typically this is an executable program, but it could be any file. A legitimate file must be selected even if this field is not actually used.

In the **Validator Command Line** textbox the Contest Administrator enters the command to be used by PC<sup>2</sup> to invoke the validator. This command can include *parameter substitutions* similar to those allowed when configuring languages. As when defining languages, parameter substitutions are indicated by a set of matching curly braces with a colon as the first character and containing a substitution keyword, for example, **{:infile}**. The following table shows the substitutions which will be performed by PC<sup>2</sup> prior to invoking the validator:

---

<sup>42</sup> In fact *gvim* is a program which exists on many systems and operates exactly like this. However, for the sake of this example, consider it to represent any arbitrary program; you don’t need to know anything about *gvim* to understand the example.

<b>Keyword</b>	<b>Meaning</b>
<b>validator</b>	Represents the file name given in the <b>Validator Program</b> box. Note that it is not a required that the “validator program” actually be an executable program; it could for example be a text file.
<b>infile</b>	Represents the problem data input file as configured in the problem.
<b>outfile</b>	Represents the output sent to <i>stdout</i> by the team program when it was executed by the judge.
<b>ansfile</b>	Represents the judge’s answer file as configured in problem.
<b>resfile</b>	Specifies the name of the file into which an active validator must place an XML representation of the judgement.

Thus to invoke *gvim* as described above the proper **Validator Command Line** entry would be:

```
{:validator} -d {:outfile} {:ansfile}
```

This would invoke the *gvim* program ( **{:validator}** ) with a “-d” argument, passing it the team program output ( **{:outfile}** ) and the judge’s correct-answer file ( **{:ansfile}** ).

Note that while the above example uses an existing program (*gvim*) as the “validator program”, it is not a requirement that this actually be an executable program. As an example, the “validator program” file might be a text file containing a set of “rules” describing how to determine the correctness of a team’s output, and a completely separate program could be invoked via the **Validator Command Line** entry, perhaps passing the “validator program” file as a parameter.<sup>43</sup>

For example, suppose a contest problem required teams to write a program to generate output which conformed to a set of lexical rules. If there existed a program named “*analyze*” which performed lexical analysis of the contents of a file according to a set of rules specified in another file, then the Contest Administrator might create an appropriate set of rules (the rules to which the team program’s output must conform) in a file named **rules.dat**, select **rules.dat** as the “**Validator Program**” file, and then specify the following as the **Validator Command Line** entry:

```
analyze {:validator} {:outfile}
```

This would invoke *analyze* as the program to be executed during the validation step, passing it **rules.dat** and the team output file as input. (What *analyze* would do with this is left as an exercise for the reader – but see the following section.)

---

<sup>43</sup> It is arguable that a better name for this feature would have been “**Validator File**” rather than “**Validator Program**”, since it does not have to be a program.

## **F.5 Implementing a Validator**

The use of an existing system tool (e.g. *gvim*) as a validator has the drawback that such validators typically must be *passive*, since they normally have knowledge of neither how to make decisions regarding correctness of team program output nor how to return such decisions to PC<sup>2</sup> even if they did know how to make them. The built-in PC<sup>2</sup> validator can be used as an *active* validator since it knows how to make decisions and knows how to return them, but as mentioned previously it is an extremely weak implementation.

Thus, it is desirable to have a mechanism which allows the use of a special-purpose program written specifically to be an active validator. In order to support the use of such validators, it is necessary to define a set of standards for interfacing validators to the system. Because a validator is a problem-specific (not contest-specific or contest control-system specific) entity, ideally these standards should be uniform and general enough that a conforming validator (and its corresponding contest problem) could also be used in conjunction with contest control systems other than PC<sup>2</sup>.

Under the auspices of the ICPC, the PC<sup>2</sup> development team has worked with a number of other contest system development teams to define such a standard. Known as the **ICPC Validator Interface Standard**, it specifies two interfaces: the invocation interface from a contest control system to a validator program, and the result interface from the validator back to the contest control system.<sup>44</sup>

The following subsections give a brief summary of the interfaces defined by the standard,<sup>45</sup> followed by a description of the PC<sup>2</sup> implementation of the standard. Since PC<sup>2</sup> complies with the standard, any validator written to comply with these interfaces can be used as an active validator for a problem in PC<sup>2</sup>.

### **F.5.1 Invocation Interface**

The standard mandates that the contest control system is responsible for invoking the validator and passing it at least four command line parameters, as follows:

**parameter1:** a string specifying the name of the input data file which was used to test the program whose results are being validated.

**parameter2:** a string specifying the name of the output file which was produced by the program being validated when it was run using the data file named in parameter1 (that is, the name of the file containing the output to be 'validated').

**parameter3:** a string specifying the name of an arbitrary "answer file" which acts as input to the validator program. The answer file may, but is not necessarily

---

<sup>44</sup> <http://www.ecs.csus.edu/pc2/doc/valistandard.html>

<sup>45</sup> The interface descriptions contained here are necessarily terse; see the standard for complete details.

required to, contain the "correct answer" for the problem. For example, it might contain the output which was produced by a "Judge's Solution" for the problem when run with the data file named in parameter1 as input. Alternatively, the "answer file" might contain information, in arbitrary format, which instructs the validator in some way about how to accomplish its task.

**parameter4:** a string which specifies the name of the "result file" which the validator must produce. The content of the result file produced by the validator is defined in the following section.

The requirements for passing parameters to a validator can be met by the Contest Administrator in PC<sup>2</sup> through the use respectively of the **{:infile}**, **{:outfile}**, **{:ansfile}**, and **{:resfile}** command substitution parameters in the **Validator Command Line**; PC<sup>2</sup> will automatically insert the appropriate values for the problem when the validator is invoked. Also, as required by the standard, the data file, program output file, and the answer file are in the current directory when the validator program is run. These conditions taken together specify how an arbitrary validator program should expect to be invoked by PC<sup>2</sup>.

The standard specifies that the contest control system may pass additional command line parameters to a validator, as long as the first four command line parameters are specified as listed above, and that the interpretation of any such parameters is up to the validator. The Contest Administrator in PC<sup>2</sup> can pass arbitrary additional parameters to the validator by including them in the **Validator Command Line** after the four required parameters.

## **F.5.2 Result Interface**

The standard requires that the validator result be returned in the "result file" whose name is specified by **parameter4** (above), and that the contents of the result file must be a valid "XML Document". This means that it must start with a valid XML declaration,<sup>46</sup> such as

```
<?xml version="1.0"?>
```

The root element of the XML document must be of the form

```
<result outcome = "string1"> string2 </result>
```

The tag name "**result**" is fixed and required by the standard, as is the attribute name "**outcome**".

"**string1**" is an "outcome string" defining the result (outcome) which the validator is reporting to the contest control system. If the value of "**string1**" is "accepted" (or any case-variation of that word), the standard specifies that the validator is indicating that the program output file "passed" the validation test(s). If "**string1**" contains any value other than a form of the word "accepted", the standard specifies that the validator is indicating that the program output file "failed" the validation test(s).

---

<sup>46</sup> Strictly speaking, the XML standard does not require that a document contain an XML header to be a valid XML document. However, the current PC<sup>2</sup> implementation expects a validator result file to have an XML header.



In PC<sup>2</sup>, the appearance of any form of the word “accepted” in the “**string1**” attribute of the result element in the result file causes PC<sup>2</sup> to assign a recommendation of “YES” to the run being executed. Recommendations are displayed to the judge if the SVTJ checkbox has been checked when the validator is configured with the contest problem.

If the value of “**string1**” is not some form of the word “accepted”, then PC<sup>2</sup> compares the actual string value to the “reject messages” originally defined in the **reject.ini** file. If “**string1**” matches one of the reject messages, then PC<sup>2</sup> assigns that message as the recommendation for the run being executed; otherwise, it assigns a recommendation of “Undetermined” to the run.<sup>47</sup>

“**string2**” is an arbitrary message string being returned from the validator to the contest control system. The standard specifies that the interpretation of this string is up to the contest control system. PC<sup>2</sup> does not use the “**string2**” parameter from the result file.

### **F.5.3 PC<sup>2</sup> Extensions**

The standard specifies that the XML **<result>** element produced by the validator may include other attributes in addition to the “outcome” attribute, and may also include additional (nested) elements; it also specifies that the interpretation of any such additional attributes and/or elements is up to the contest control system. Such additional attributes can be used to implement a variety of features.

PC<sup>2</sup> makes use of additional attributes to implement a form of security. Specifically, it expects the validator to define an additional attribute named “**security**” and to return in that attribute *the name of the result file*. That is, PC<sup>2</sup> expects the XML result file to look like:

```
<?xml version="1.0"?>
<result outcome="string1" security="resfile"> string2 </result>
```

where “**resfile**” is the value which was passed to the validator as the name of the file into which the results should be placed (and where **string1** and **string2** are as described above).

Each time PC<sup>2</sup> invokes a validator it generates a unique random name for the result file. When the validator returns PC<sup>2</sup> examines the contents of the result file and verifies that the **security** attribute value matches the file name. Since a user (team) program cannot know ahead of time what result file name PC<sup>2</sup> will generate, it is not possible for a user program to generate a “fake” result file which somehow gets used in place of one generated by the validator. While this is not a complete guarantee of security, it does make it much more difficult for a user program to circumvent the operation of the validator.

---

<sup>47</sup> Note: as mentioned previously, PC<sup>2</sup> appends the five characters “No - ” (that is, the word “No” followed by a space, a hyphen, and another space) to each “reject” message found in the **reject.ini** file. Therefore the strings contained in the “**string1**” attribute returned by a validator should also contain those characters.

## Appendix G – Language Definitions

As described earlier in this manual, PC<sup>2</sup> must be given a “language definition” for each language to be used in the contest (that is, for each tool which teams can use to write and submit programs). The language definition consists of four distinct text strings: the “Display Name”, the “Compile Command Line”, the “Executable Filename” specification, and the “Program Execution Command Line”.

In order to help in understanding how such language definitions work (and so that you will be better able to develop your own language definitions), it is useful to understand what it is that PC<sup>2</sup> *does* with a language definition. Language definitions are used by PC<sup>2</sup> in two circumstances: when a Team invokes a **TEST RUN** operation, and when a Judge or an Admin invokes an **EXECUTE** operation. The following algorithm describes the sequence of steps which PC<sup>2</sup> follows when either the **TEST RUN** button on the Team, or the **EXECUTE** button on the Judge or Admin is pressed.

1. The entire contents of the “**execute**” directory (beneath the \$PC2HOME directory) are deleted. If something prohibits this clearing, the system stops and displays a warning message, and all remaining steps are skipped.
2. The submitted files are copied to the **execute** directory.
3. If the file whose name is specified as the “Executable Filename” in the language definition exists in the **execute** directory, it is deleted. This prohibits a team from submitting an executable file (or more correctly, they can submit it but it will never be executed).
4. The command specified as the “Compile Command” in the language definition is executed, using appropriate command parameter substitutions as defined earlier in this manual.
5. PC<sup>2</sup> checks for the existence in the **execute** directory of a file whose name matches the specified “Executable Filename”. If this file exists, it must have been created by the execution of the “Compile Command”. (This is how PC<sup>2</sup> determines whether compilation was successful.)
6. If the specified “Executable Filename” exists (hence, the “Compile Command” was successful), then the following operations are performed:
  - a. The data file associated with the problem (if any) is copied into the **execute** directory.
  - b. The command specified as the “Program Execution Command Line” is executed, using appropriate command parameter substitutions as defined earlier in this manual.
  - c. If this is an EXECUTE operation on a Judge or Admin (as opposed to a TEST RUN on a Team), then if there was a “Validator”

associated with the problem, then the following operations are performed:

- i. The “answer file” associated with the problem (if any) is copied into the **execute** directory.
- ii. The command specified as the “Validator Command Line” is executed, using validator command parameter substitutions as defined earlier in this manual (see the Appendix on Validators).
- iii. If “Show Validator Result To Judge” (SVTJ) was checked when the Validator was associated with the problem, PC<sup>2</sup> reads the result file created by the Validator (see the Appendix on Validators) and displays the appropriate result for the Judge.

Some examples of PC<sup>2</sup> language definitions which have been used in past contests and were known to work in those environments are shown below. Each definition consists of four lines, corresponding to the four text field entries required on the Edit Language screen when defining a new language under the main Administrator screen.

No guarantee is made that these definitions will work in *your* environment, nor that they will not become obsolete due to changes made by the various language tool vendors. All we can tell you is that all of these language definitions have been used successfully in past contests. Use them at your own risk.

#### Language: Java

```
Java
javac {:mainfile}
{:basename}.class
java {:basename}
```

#### Language: GNU C++

```
GNU C++
g++ -lm -o {:basename} {:mainfile}
{:basename}
./{:basename}
```

## Language: GNU C

```
GNU C (gcc)
gcc -lm -o {:basename} {:mainfile}
{:basename}
./{:basename}
```

## Language: Borland Kylix C++

```
Kylix C++
bc++ -A {:mainfile}
{:basename}
./{:basename}
```

## Language: Borland Kylix Delphi (Pascal)

```
Kylix Delphi (Pascal)
dcc {:mainfile}
{:basename}
./{:basename}
```

## Language: Microsoft Visual C++

```
MSVC++
cl {:mainfile}
{:basename}.exe
{:basename}.exe
```

## Language: Turbo Pascal for DOS

```
Turbo Pascal
tpc {:mainfile}
{:basename}.exe
{:basename}.exe
```

One of the ramifications of the sequence of language-handling steps described above is that a team cannot submit a program whose file name is the same as the “Executable Filename” specified in the corresponding language definition. For example, if the Contest Administrator configured a language by saying that the result of a compile operation for the language was to produce an executable file whose name was always “**a.out**”, then if a team submitted a *source code program* in a file named “**a.out**”, then the source code program file would get deleted (step 3) prior to the compile step.

Normally this difficulty is eliminated through the use of command parameter substitutions; the Contest Administrator would not normally specify “**a.out**” as the expected executable file to be generated by the compilation steps, but rather would use a specification such as

“{:basename}.out”, and further a team would normally submit source code in a file named, e.g. “a.c” rather than “a.out”.

However, there is one scenario under which the mechanics of language handling by PC<sup>2</sup> can cause difficulties (or at least, confusion). This is the case of purely interpreted languages, such as Perl or shell-script. In these cases there is no “compilation” step which is expected to generate an “executable” file; the “source file” is effectively the same as the “executable” file (in the sense that the source file undergoes no transformation prior to invoking “execution”, since “execution” involves running an interpreter against the original source program).

For example, in the case of Perl, the Contest Administrator might attempt to configure the language definition as:

```
Perl
/bin/perl -c {:mainfile}
{:mainfile}
/bin/perl {:mainfile}
```

This definition says that the language Display Name is “Perl”; that the “Compile Command” invokes `/bin/perl` (the Perl interpreter) with the “-c” (check syntax) argument and the submitted file, that the result of “compilation” is to produce an “executable” file whose name is the same as the submitted file, and that following the “compilation” step PC<sup>2</sup> should check for the existence of the submitted file and if present it should invoke `/bin/perl` again, this time executing the Perl commands in the submitted file.

However, this language definition will not work, because of the steps which PC<sup>2</sup> follows: it will delete the submitted source code (`.pl`) file prior to invoking the compilation command. Again, the reason for this is that it checks for the existence of the specified “executable file” *after* the compilation step, and assumes that if the file exists then the compilation was successful. Thus if a team submitted a source file named “myFile.pl”, since the submitted file is the “executable file” which would be input to the Perl interpreter, the “executable filename” (after command parameter substitution) would also be “myFile.pl” – but the file would have been deleted.

It is still possible to use such languages with PC<sup>2</sup>. The trick is to create a separate “script” file which acts as the “Compile Command” and has the effect of creating a separate file which has the same name as that specified for the “Executable Filename” and which PC<sup>2</sup> can test for after compilation and prior to invoking program execution.

For example, suppose the Contest Administrator creates a shell script file named “compilePerl” with the following contents (the example presumes a Unix-like environment, but a similar approach can be taken in a Windows system):

```
#!/bin/csh
perl -c $*
if ($? == 0) then
    touch OK
endif
```

This script basically says: run the “C-Shell” interpreter (line 1); have it execute the perl interpreter and perform a syntax check (-c) on the arguments passed to the script (\$\*) (line 2);

check the system “status variable” (\$?) and if it is zero (meaning no errors occurred) (line 3) then create a file named “**OK**” (line 4).

With this `compilePerl` script accessible via the PATH variable, the following language definition will allow a Perl program to be submitted and processed by PC<sup>2</sup>:

```
Perl
compilePerl {:mainfile}
OK
perl {:mainfile}
```

This language definition will invoke the `compilePerl` script telling it to syntax-check the submitted program file, then if the file “OK” exists (which will only happen if the Perl syntax-check was successful) it will invoke the Perl interpreter to execute the submitted program. Note that while this example is for Perl, other languages such as Bourne Shell (and other shells), Python, Ruby, and ‘awk’ can also use a similar solution.

The above example should provide some insight into the types of operations which the Contest Administrator can invoke from PC<sup>2</sup>. For example, it is possible to create a script file which is invoked for the “Program Execution Command” and does any desired operation, such as copying a data file into the `execute` directory prior to running the intended program. Basically any desired operation can be performed at either the “compile” or “execute” step, as long as one has a clear understanding of the PC<sup>2</sup> language processing algorithm described above. This organization of language processing gives a great deal of flexibility to the Contest Administrator.

## Appendix H – Extending PC<sup>2</sup>

While the client interfaces (Admin, Judge, Team, and Board) in PC<sup>2</sup> are intended to be as general as possible, there may be situations where users would like a client to operate differently. For example, a user may wish to create a scoreboard that uses a different scoring algorithm, or to create a different sort of contest system interface for Teams. To support this, PC<sup>2</sup> provides a mechanism for users to create their own “custom clients” which interface with the rest of the PC<sup>2</sup> system.

All clients in PC<sup>2</sup> are extensions of a Java class named **BaseClient**.<sup>48</sup> The **BaseClient** class defines an interface between the PC<sup>2</sup> server and an arbitrary graphical user interface (GUI). As a result, any class which extends **BaseClient** can be built to provide any desired GUI, and can use the methods in **BaseClient** to obtain access to the PC<sup>2</sup> server.

Because **BaseClient** is a general-purpose interface to the server, it contains many methods which are not needed (or desirable to have) in certain kinds of clients. To further simplify the process of creating custom clients, PC<sup>2</sup> defines three abstract subclasses of **BaseClient**: **TeamClient**, **JudgeClient**, and **ScoreboardClient**. Each of these abstract classes defines specifications (prototypes) for the methods which a client of the corresponding type needs to implement. Thus for example a user interested in creating a new type of Team client should extend the abstract **TeamClient** class (rather than directly extending **BaseClient**), and implement the abstract methods specified in **TeamClient**.

The **BaseClient** class, along with all its direct and indirect subclasses, is part of a Java package named “**ex**” (“extensions”) which in turn is a subpackage of package “**pc2**” (thus the full class name for **BaseClient** is **pc2.ex.BaseClient**). The downloadable PC<sup>2</sup> distribution contains a directory named “doc” in which exists a subdirectory named “**pc2ex**” containing the “Javadoc” documentation for the entire “**pc2.ex**” package; opening a browser on the file “**index.html**” in the `$PC2HOME/doc/pc2ex` directory will display the documentation. Users who are interested in additional information on developing custom clients should contact us through the PC<sup>2</sup> web page for further details.

---

<sup>48</sup> Strictly speaking, in the present version of PC<sup>2</sup> the Admin client is not an extension of **BaseClient**; however, there is no reason in principle that it could not be. All the other PC<sup>2</sup> clients are literally subclasses of the **BaseClient** class.

## Appendix I – Frequently Asked Questions

This appendix covers some of the common questions we get. For additional information, visit the PC<sup>2</sup> web page FAQ at <http://www.ecs.csus.edu/pc2/doc/faq/index.html>.

**Question:** Why can't my PC<sup>2</sup> clients connect to the PC<sup>2</sup> server when I run the server on my Linux machine? (It works fine when the server is running under Windows, even if the clients are on Linux machines.)

**Answer:** We have seen this happen when a Linux machine running the PC<sup>2</sup> server has not been properly configured for a network environment. Specifically, Linux machines typically start up configured to report their "machine name" as "localhost". PC<sup>2</sup> queries the OS to obtain the machine name, and then maps this name to a corresponding IP address. The machine name "localhost" maps to the "loopback" IP address 127.0.0.1. This ends up being the IP address which the PC<sup>2</sup> server broadcasts to the clients as the address they should use for communication back to the server -- which means the clients will be unable to contact the server because they will use their own local loopback channel.

To check whether your Linux machine running the PC<sup>2</sup> server has this problem, type the command "hostname" (or "/bin/hostname") at a command prompt. If this returns "localhost", your OS is not properly configured for network operations.

One way to fix this problem on most machines is to edit the file **/etc/sysconfig/network** and change the HOSTNAME entry to a host name which resolves to a non-local IP address, and then reboot the machine.

Another approach is to use the "hostname" command, run as root, to set the hostname to either a specific non-local IP address or to a hostname which resolves to a non-local IP address. One user with this problem wound up changing the hostname before starting PC<sup>2</sup>, via a command line like:

```
hostname $(ifconfig eth0 | grep inet | sed 's/.*r:\| B.*//g')
```

or (depending on your shell)

```
hostname `ifconfig eth0 | grep inet | sed 's/.*r:\| B.*//g'`
```

which is basically getting the address from the eth0 interface and setting the hostname to that value. In any case, be sure to coordinate such actions with your local network administrator.



**Question:** What does it mean when the system generates the message "NoClassDefFound" and won't run ?

**Answer:** This is a Java error message meaning that it could not find one of the "classes" (programs) required to run the system. The missing classes could either be PC<sup>2</sup> modules or Java system modules. There are a number of possible causes for this error. Here are some things you can try:

Verify that the PC<sup>2</sup> system was unzipped "including subdirectories" and "with case-sensitivity". PC<sup>2</sup> uses many Java "packages", which must be stored in the proper subdirectories beneath the PC<sup>2</sup> installation directory. Make sure the program which was used to "unzip" the distribution properly created and maintained the hierarchical directory structure and the "case" of filenames in the system.

Verify that the CLASSPATH environment variable includes the Java/lib and PC<sup>2</sup> installation directories.