INTERNATIONAL COLLEGIATE
PROGRAMMING CONTEST

California State University, Sacramento's

PC²

Version 9.7

Contest Administrator's Installation and Configuration Guide

<Last Update: March 14, 2021>

Table of Contents

1 Introduction 5
1.1 Overview 5
1.2 Compatibility Note 6
1.3 References 6
2 Getting Started 7
2.1 Server Startup 7
2.2 Admin Startup 7
2.3 Contest Configuration 8
2.4 Team Startup 8
2.5 Judge Startup 9
2.6 Scoreboard Startup 9
2.7 Starting the Contest 10
2.8 Additional Information 10
3 Installation Details 12
3.1 Installation 12
3.2 Network / Firewall Requirements 12
3.3 Memory Limits 13
3.4 Security Alerts 14
3.5 Uninstall 14
4 PC² Initialization Files 16
4.1 The *pc2v9.ini* file 16
4.2 Other Initialization Files 18
5 PC² Startup Procedures 19
5.1 Built-in Commands 19
5.2 Server Startup 20
5.2.1 Non-GUI Server Startup 22
5.3 Server GUI Controls 22
5.3.1 Adding Sites 23
5.3.2 Restarting / Reconnecting Servers 24
5.3.3 Connections and Logins 26

5.3.4 Additional Server GUI Controls 26
5.4 Starting Clients 26
5.5 Contest Profiles 27
6 Interactive Contest Configuration 32
6.1 Administrator Login 32
6.2 User Accounts 33
6.2.1 Account Creation 33
6.2.2 Account Names and Passwords 34
6.2.3 Loading Account Data 37
6.2.4 Importing ICPC Data 38
6.3 Contest Problems 39
6.3.1 Defining a Problem 39
6.3.2 Multiple Test Data Files 43
6.3.3 Defining Judging Type 45
6.3.4 Assigning Auto Judging to Judge modules 46
6.4 Contest Languages 50
6.4.1 Defining a Language 50
6.4.2 Command Parameter Substitutions 53
6.4.3 Language Definition Examples 53
6.4.4 Language Definitions In Multi-Site Contests 55
6.5 Contest Judgments 57
6.5.1 Defining a New Judgment 57
6.5.2 Changing Existing Judgments 58
6.6 Balloon Notifications 59
6.6.1 Defining Balloon Notifications 60
6.6.2 Email Server Advanced Settings 61
6.7 Options (Settings tab) 62
6.8 Sites 65
7 Configuring the Contest via Configuration Files 66
7.1 Loading Configuration Files via the PC² Server 66
7.2 Loading Configuration Files via the PC² Admin 67
7.3 Additional Configuration File Capabilities 67
8 Starting the Contest 69
8.1 Clock Control 69
8.1.1 Starting the Contest Manually 69
8.1.2 Starting the Contest Automatically 70
8.2 Contest Length 71

8.3 Multi-Site Clock Control 72
8.4 Practice Sessions: Resetting A Contest 74

9 Monitoring Contest Status 76
9.1 Team Startup Status 76
9.2 The Runs Display 77
9.3 Editing Runs 78
9.3.1 Extracting Runs 80
9.4 Filtering Runs 81
9.5 Clarifications 82
9.6 Reports 83
9.6.1 Automatic Generation of Reports at End of Contest 83
9.7 Event Feed 85
9.8 Web Services 85

10 The PC² Scoreboard 86
10.1 Overview 86
10.2 Scoring Algorithm 86
10.3 Configuring Scoring Properties 87
10.4 Starting the Scoreboard 88
10.5 Scoreboard Updates 90
10.6 Scoreboard HTML Files 90
10.7 Scoring Groups 92
10.8 Managing HTML File Generation 93
10.9 No-GUI Mode 95

11 Finishing the Contest 96
11.1 Finalizing 96
11.2 Exporting Contest Results 97
11.2.1 Generating a *results.tsv* export file 97
11.2.2 Generating a *pc2export.dat* export file 97
11.3 Shutting Down 97

Appendix A – pc2v9.ini Attributes 99

Appendix B – Networking Constraints 101

Appendix C – PC² Server Command Line Arguments 103

Appendix D – ICPC Import/Export Interfaces 105

Appendix E – Output Validators 111

Appendix F – Language Definitions 124

Appendix G – Using the PC² API 129

Appendix H – Troubleshooting / Getting Help 130

Appendix I – PC² Distribution Contents 131

Appendix J – Log files 132

Appendix K – Reports Program 133

Appendix L – PC² XML (Legacy) Event Feed 136

Appendix M – PC² Web Services 140

Appendix N – PC² Team Clients 143

Appendix O – Input Validators 146

Appendix P – reject.ini 153

Appendix Q – GUI Customization 155

Appendix R – Shadow Mode 157

1 Introduction

1.1 Overview

PC^2 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^2. Further information on PC^2, including how to obtain a copy of the system, can be found at http://pc2.ecs.csus.edu/ .

PC^2 operates using a client-server architecture. Each site in a contest runs a single PC^2 ***server***, and also runs multiple PC^2 ***clients*** which communicate with the site server.[1] Logging into a client using one of several different types of PC^2 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^2 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^2: (1) a machine running a PC^2 server must be able to communicate via TCP/IP with every machine running a PC^2 client at its site; and (2) in a multi-site contest, every machine running a PC^2 server must either be able to communicate via TCP/IP with the machines running PC^2 servers at every other site or else must indicate that it requires "proxy support" from another server. In particular, for servers which do not request proxy support, there must not be any firewalls which prohibit communication with other servers; the system will not operate if this communication is blocked.[2] It is not necessary for client machines to be able to contact machines at other sites.

Each PC^2 module (server or client) reads one or more 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^2 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^2 initialization files in their account.

---

[1] Note that "site" refers to a logical grouping of contest participants, not (necessarily) a physical or geographic grouping. It is perfectly possible for participants at physically or geographically separate locations to collectively represent one PC^2 "site" and to be served by a single "site server"; it is quite common to run an entire contest using a single site server regardless of whether or not the teams, judges, etc. are physically close. Multi-site operations are primarily supported for situations where network connectivity is highly unreliable.

[2] See the Appendix titled "Networking Constraints" for further details on using PC^2 over networks.

Setting up and running a contest using PC^2 involves the following steps: (1) installing Java and PC^2 on the contest machines; (2) creating/editing the necessary initialization files; (3) starting the server(s) and clients(s); (4) configuring PC^2 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.

1.2 Compatibility Note

Starting with Version 9.3, the PC^2 system contains substantial new enhancements which are incompatible with earlier versions of the system. Users should not "mix" components of PC^2 V9.3+ with earlier versions (9.2.x and below).

1.3 References

While this manual tries to give a complete description of installing and using PC^2, the following web references may provide additional helpful information. In particular, the PC^2 Wiki is updated much more frequently than this manual and should be consulted frequently for answers to questions.

PC^2 home page
http://pc2.ecs.csus.edu/

PC^2 Wiki – up to date articles and information about PC^2
http://pc2.ecs.csus.edu/wiki/Main_Page

PC^2 Bugzilla - enhancement and defect tracking and reporting for PC^2
http://pc2.ecs.csus.edu/bugzilla/

2 Getting Started

For those people who hate to read manuals and would rather take a chance with a shortcut list, this chapter provides a (somewhat terse) summary of the steps necessary to install PC^2 and get it running for your contest. Please note that this chapter does not provide all the details; it is intended to give readers a general overview of the process of using PC^2 to run a contest. Please refer to the subsequent chapters for necessary additional details.

2.1 Server Startup

Each contest site is controlled by a "site server" (or just "server" for short).[3] Perform the following steps to install and start a PC^2 site server (see the chapter on Installation Details, as well as the chapter on Startup Procedures and the Appendices, for further details).

- Install Java (version 1.8 or greater) on the machine which will act as your PC^2 Server; ensure the Java ***bin*** directory is in the PATH.
- Download the latest PC^2 distribution from https://pc2.ecs.csus.edu/. Install PC^2 by unzipping the PC^2 distribution to a directory of your choice.
- Add the PC^2 *bin* directory to the PATH.
- Start a PC^2 server using the command "**pc2server**".
  - Login using the name "site1" and password "site1".
  - Enter a contest password (this contest security password will be required to be reentered for all subsequent server startups).

2.2 Admin Startup

Overall contest management is controlled through the PC^2 "Administrator Client" or "Admin" for short. Perform the following steps to get the PC^2 Admin running.

- If you are going to run the PC^2 Admin on a different machine than the one running the PC^2 Server, then on the Admin machine follow the above steps regarding installing Java, downloading/unzipping the PC^2 distribution, and setting the PATH values.
- If the Admin is on a separate machine from the Server, edit the **pc2v9.ini** file on the Admin machine, replacing "**localhost**" in the **[client]** section with the IP address of the Server machine.
- Start the PC^2 Admin using the command "**pc2admin**" and login using the name "root" and password "administrator1"

[3] See the previous footnote regarding the definition of "contest site".

2.3 Contest Configuration

Use the PC^2 Admin to configure at least the following contest items by selecting the correspondingly-named tab on the Admin **Configure Contest** screen: [4]

- **Accounts**: generate the necessary accounts – minimally, you'll need to generate one account for each team, one or more judge accounts, and a scoreboard account (see the section on *User Accounts* in the chapter on *Interactive Contest Configuration* for further details).

  You may also want to generate additional judge accounts for "auto-judging" (see the section on *Contest Problems* in the chapter on *Interactive Contest Configuration* for further details), and you may want to generate a "feeder" account for external tools accessing PC^2 (see the *Event Feed* and *Web Services* sections in the chapter on *Monitoring Contest Status* for further details).

- **Problems**: define one or more contest problems, specifying the problem input data file(s) (if the problems require input data), the judging type, and the "validator specification" (if the problems are to be "auto-judged"). See the section on *Contest Problems* in the chapter on *Interactive Contest Configuration* for further details.

- **Languages:** define one or more contest languages, specifying the language name, compile command line, executable filename, and program execution command line. See the section on *Contest Languages* in the chapter on *Interactive Contest Configuration* for further details.

Note that all configuration settings can be specified either interactively or by creating and loading a "YAML Configuration File". See the chapters on *Interactive Contest Configuration* and *Configuring the Contest via Configuration Files* for information on both these options.

2.4 Team Startup

There are two methods which can be used to allow teams to login to a PC^2 contest: the PC^2 Team Application client and the PC^2 Team Web client. Using the Team Application client requires installing PC^2 on each team machine, whereas using the Team Web client allows teams to use a browser to connect to the contest but requires providing an external web server and installing the PC^2 Team web pages into the web server.

If you are going to use the PC^2 Team Application client, perform the following steps to install the client on each team machine:

- Follow the above steps regarding installing Java, downloading/unzipping the PC^2 distribution, and setting the PATH values on each team machine.

[4] Starting with Version 9.7, PC2 also supports user-customization of the images on the login screen; see the Appendix on *GUI Customization* for details.

- Edit the **pc2v9.ini** file on each team machine, replacing "**localhost**" in the **[client]** section with the IP address of the PC2 Server machine.

- Start the PC² Team client using the command "**pc2team**" and login using the team name and password defined under "**Accounts**", above. (The default team name and password are both "teamxx", where "xx" is the team number.)

To use the PC² Team Web client, see the Appendix on PC² Team Clients.

2.5 Judge Startup

There are two methods which can be used for judging team submissions: *manual* and *automated* (the two methods are not mutually exclusive; automated judging can be used to send teams a "preliminary result" and can then be followed by manual [human] judging). Selecting the type of judging is part of Contest Problem Configuration, described above and also in more detail in the section on *Contest Problems* in the chapter on *Interactive Contest Configuration*. In any case you will want to set up one or more judge machines.[5]

Perform the following steps to install the PC² Judge client on each judge machine:

- Follow the above steps regarding installing Java, downloading/unzipping the PC² distribution, and setting the PATH values on each judge machine.

- Edit the **pc2v9.ini** file on each judge machine, replacing "**localhost**" in the **[client]** section with the IP address of the PC² Server machine.

- Start the PC² Judge client using the command "**pc2judge**" and login using the judge name and password defined under "**Accounts**", above. (The default judge name and password are both "judgexx", where "xx" is the judge number.)

If your contest is going to use automated judging, see the section *Assigning Auto Judging to Judge Modules* in the chapter on *Configuring the Contest*.

2.6 Scoreboard Startup

Perform the following steps to install the PC² Scoreboard client on a machine[6]:

- If you are running the Scoreboard client on a different machine (one onto which PC² has not already been installed), follow the above steps regarding installing Java,

[5] In principle judging can be done on the same machine used to run the PC² Server or the PC² Admin, but we don't recommend it because of the problem of contest administrators and judges colliding with each other trying to use the machine.

[6] Unlike the situation with judges, there only needs to be a *single* PC² Scoreboard for the entire contest and it rarely needs attention once it is started; for this reason we frequently run the scoreboard on the same machine as the PC² Server or Admin.

downloading/unzipping the PC^2 distribution, and setting the PATH values on the scoreboard machine.

- Edit the **pc2v9.ini** file on the scoreboard machine, replacing "**localhost**" in the **[client]** section with the IP address of the PC^2 Server machine.

- Start the PC^2 Scoreboard client using the command "**pc2board**" and login using the scoreboard account name and password defined under "**Accounts**", above. The default scoreboard name and password are both "boardX", where "X" is the number of the scoreboard account (typically "1").

- Arrange for the scoreboard-generated HTML files to be accessible to user's browsers. See the section *Scoreboard HTML Files* in the chapter on the *PC^2 Scoreboard* for further details.

2.7 Starting the Contest

The contest can be started either manually, or automatically at some scheduled time. The length of the contest can be configured, and contests can be "reset" (e.g., between Practice and Real contests). See the chapter on *Starting the Contest* for additional information.

- To start the contest manually, press the "**Start ALL**" button on the Administrator module **Times** tab. This will allow teams to submit runs, judges to fetch and judge the runs, and standings to be posted on the scoreboard, and will use the PC^2 default contest length of 5 hours.

2.8 Additional Information

The above will hopefully provide enough information to allow you to get a contest going under PC^2. Note however, that it omits *many* details, including alternative configuration options, additional features, and so forth. For example, it does not take into account any of the following PC^2 capabilities:

- Use of external interfaces such as the Event Feed and PC^2 Web Services
- Command-line submission of runs by teams
- Generation of PC^2 Reports describing system information
- Monitoring team status during the contest
- Examining/editing/judging runs during the contest
- Importing contest configuration information from data files
- Exporting final results to other systems
- Configuring multi-site contests
- Creating different "scoring groups" on the Scoreboard

- Creating customized HTML scoreboard output
- Running PC^2 modules in "no-gui" mode
- Using "Profiles" to switch between contest configurations
- Handling security alerts
- Details of configuring contest problems (things like validators for automated judging, handling very large data files, etc.)
- Contest language configuration details
- Defining judgement messages
- Sending "balloon notifications" for correct submissions
- Configuration options such as maximum allowed team output, information displayed to judges, editing scoring properties, etc.
- Programmatic access to PC^2 via the PC^2 API

These things and many more are covered in the rest of this manual, and also on our Wiki at https://pc2.ecs.csus.edu/wiki.

3 Installation Details

In the event that the preceding chapter is a bit too terse, the remainder of this manual discusses the details of using PC^2 to configure and run a contest. The first step is to install the necessary software, as described in this chapter. The remaining chapters cover initialization files, starting the system, configuring the system for a contest, starting and monitoring the contest, and using the PC^2 scoreboard. In addition several appendices cover details of certain topics.

3.1 Installation

1. Install the Java Standard Edition (SE) Software Development Kit (SDK) or Java Runtime Environment (JRE), version 1.8 or later on each machine. The remainder of this manual assumes that "$JAVAHOME" represents the SDK installation directory. **We recommend using 64-bit Java, in particular for the PC^2 Server and (if used) the Web Team Interface client.**

2. Ensure "$JAVAHOME/bin" is contained in the PATH environment variable on each machine (i.e., for each user).

3. Go to the PC^2 home page (see Chapter 1, under References), navigate to the "Download" page, and download the latest PC^2 "**.zip**" or "**.tar.gz**" file to the directory where you wish to install PC^2 (this can be any directory of your choice).

4. Unzip the downloaded file, being sure to tell the unzip program to "retain directory hierarchy" and "preserve case sensitivity". This will create a directory named (for example) **pc2-9.7.0**, which for the remainder of this manual we refer to as the $PC2HOME directory. The $PC2HOME directory contains **bin**, **lib**, **doc**, **samps** and other directories, plus a default "**pc2v9.ini**" file (see the following chapter) along with several text files giving basic information such as the system version number. The **doc** and **samps** directories contain the system documentation and a variety of sample scripts, files, and other goodies you might want to examine. The file "**doc/index.html**" can be used to browse the documentation. (NOTE: all files and directories which comprise a PC^2 distribution (.zip or .tar.gz file) will unzip into the $PC2HOME directory. See the appendices for a complete description of PC^2 distribution contents.)

5. Add the PC^2 "bin" directory (that is, the directory $PC2HOME/bin) to the PATH environment variable on each machine.

3.2 Network / Firewall Requirements

Here are the PC^2 firewall requirements; see also the Appendix "Networking Constraints".

1. PC^2 Clients need to be allowed to make outbound connections to their server; hence servers need to be open to inbound connections from the clients.

2. PC^2 Servers need to be allowed to make outbound connections to all other servers.

3. PC^2 Servers either need to be open to inbound connections from all other servers, or else need to specify that another server is going to act as a "proxy" for inbound connections (see the Chapter on *PC^2 Initialization Files* and the Appendix on *Networking Constraints* for information on how to set up proxy servers).

3.3 Memory Limits

PC^2 is written in Java and therefore inherits all the features (good and bad) of that language. Although Java is touted as "run anywhere", the reality is that many things depend on your specific installation – both the version of Java you are using and the underlying OS platform on which it is running. This is particularly true when it comes to memory allocation – primarily, what Java refers to as "Heap Space".

Throughout this manual we describe various "commands" (scripts, batch files) that PC^2 provides to invoke different functions. Most of these commands involve starting a "Java Virtual Machine (JVM)". Each instance of a JVM has a "maximum heap size" value associated with it; if the Java program attempts to use more "heap space" than the maximum allowed, the program aborts with an "out of heap space" error. This can be particularly disconcerting if it is your PC^2 server (for example) that crashes in the middle of a contest. It is therefore important to know what can be done to avoid such memory-limit problems.

Each of the PC^2 commands passes a parameter to the JVM to set the maximum heap size. The form of this parameter is **`-Xmx$$$$M`**, where $$$$ represents the maximum heap size in megabytes ('M'). The heap size values used by the various scripts are conservative, intended to insure that the command will be able to run successfully on the widest variety of platforms.

On any *particular* platform it is possible that the value in the script may be significantly less that the platform is capable of supporting. It is also possible that the command value is actually *lower* than the *default* setting which would be applied on a particular platform if *no* value had been specified. This is particularly true between 32-bit and 64-bit systems.

For these reasons, you may want to determine the maximum heap size allowable on your platform and edit the corresponding script to set a larger value (this is particularly important for a large contest – one with a large number of teams, problems, and/or large data files, and is also particularly important for the PC^2 *Server*). Setting the JVM maximum heap size can significantly increase the ability of the corresponding PC^2 module to run without problems.[7]

The maximum heap size can be determined by running the following command on the corresponding platform:[8]

---

[7] The *pc2server* scripts (for both Windows and Linux) attempt to make an intelligent guess as to the type of platform (32- or 64-bit) and set the heap size accordingly. The user can change the heap size values used by these scripts by editing constants at the top of the *pc2server* script. For all the other scripts, it is necessary to edit the –Xmx parameter in the script.

[8] It may also be possible to use the parameter **–XX:+AggressiveHeap** to instruct the JVM to automatically force the heap to the largest available size. However, this parameter is non-standard, may not be available on all systems, and has not been thoroughly tested in PC^2.

Windows: **`java -XX:+PrintFlagsFinal -version | findstr HeapSize`**

Linux/MacOS: **`java -XX:+PrintFlagsFinal -version | grep HeapSize`**

3.4 Security Alerts

There are a number of security features in PC^2 Version 9, including disk file encryption, network traffic encryption and security alerts. Security alert windows will automatically appear on server UI and Admin UI when a potential security violation or exceptional condition occurs. For example, a security alert occurs if a PC^2 client logs in successfully while the same client is already logged in. Security information is also recorded in a security log in the **logs** directory. A typical security log file will be named something like: `logs/SERVER0@site1.security-0.log`. An example of the log entries is:

```
140812 155846.072|SEVERE|Thread-36|newMessage|SecurityException From:
      ADMINISTRATOR1 @ site 1 ADMINISTRATOR1 @ site 1: duplicate login
      request; previous login forced off
|edu.csus.ecs.pc2.core.exception.ContestSecurityException: ADMINISTRATOR1 @
      site 1: duplicate login request; previous login forced off
|    at edu.csus.ecs.pc2.core.InternalController.attemptToLogin (Unknown
      Source)
|    at edu.csus.ecs.pc2.core.InternalController.receiveObject (Unknown
      Source)
|    at edu.csus.ecs.pc2.core.transport.TransportManager$1.run (Unknown
      Source)
|    at java.lang.Thread.run (Thread.java:745)
140812 155846.073|SEVERE|Thread-36|newMessage|SecurityException Sec. Message:
      ADMINISTRATOR1 @ site 1: duplicate login request; previous login forced
      off  ConnHandId Socket[addr=/127.0.0.1,port=53488,localport=50002]-
      2471900182656421556
```

3.5 Uninstall

To uninstall PC^2 do the following on each PC^2 Server and client machine:

1. Use the **pc2reset** script to remove the contents of any PC^2 directories
2. Remove the **pc2v9.ini** file and the **archive** directory.
3. Remove the **$PC2HOME** directory and its contents
4. Restore the system environment variables (**PATH** and **CLASSPATH**)

PC^2 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 on a

Windows machine, nor does it copy any files to locations outside the installation directory or the current working directory in any environment.

4 PC² Initialization Files

When a PC² module (server or client) begins running, it reads an "initialization file" named **pc2v9.ini** from the directory in which it was started.[9] The Contest Administrator must ensure this file is present and edited as necessary on each machine prior to starting a PC² module.

This chapter describes the initialization files and their contents (note: some default versions of the initialization files are provided in the *samps* folder of the PC² distribution package; these must be edited as necessary). Further descriptions of initialization files and their contents can be found in the Appendices.

4.1 The *pc2v9.ini* file

Every PC² module reads a file named "**pc2v9.ini**" at startup. By default each module looks for its **pc2v9.ini** file *in the current working directory*; thus for example Team machines would typically need to have a **pc2v9.ini** file in their "home" directory. (Command-line arguments can be used to point to a different location for the **pc2v9.ini** file; see the Appendices for details.)

The **pc2v9.ini** file provides key initialization information to the PC² module. The file is formatted in sections. Each section starts with a section-name in square brackets. The following different section names are recognized: **[server]** and **[client]**. Each PC² module reads the entire file, but silently ignores any information which does not pertain to it (for example, server modules ignore data in all sections except the **[server]** section, etc.) All lines starting with "#" or ";" are comments and are also ignored, as are blank lines.

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 PC² configuration attributes. The "**value**" is the value to which the corresponding attribute is set when the **pc2v9.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 **pc2v9.ini** file. However, certain attributes *must* be specified in the **pc2v9.ini** file in order for the system to function properly.

---

[9] Older versions of PC² also read files named **reject.ini** and **sitelist.ini**. However, the functionality provided by those files is incorporated into interactive screens in the Administrator client in PC² V9, and their use is deprecated and scheduled for removal in a future version of the system.

Based on these rules, a minimum sample **pc2v9.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* **pc2v9.ini** file containing all the required entries and put this same file on all machines at a given site.

```
# sample pc2v9.ini file

[client]
# Tell PC2 clients where to find their server (IP and port)
server=198.51.100.50:50002
```

The sample **pc2v9.ini** file shown above would be appropriate for both client and server machines in a single-site contest.

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 **pc2v9.ini** file.

By default (that is, in the absence of any information in the **pc2v9.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 **pc2v9.ini** file of that 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 **[server]** section of the **pc2v9.ini** file when a server starts, the server implicitly assumes it is a primary server.

Thus for example, the **pc2v9.ini** file on the primary server in a multi-site contest might look like the sample above (since it does not contain any **remoteServer** attribute), whereas the **pc2v9.ini** file on machines at a second site might look like:

```
# sample pc2v9.ini for a second site

[client]
# tell clients where to find their site's server (IP and port)
server=203.0.113.7:50002

[server]
# Tell this (secondary) server how to contact the primary server
remoteServer=198.51.100.50:50002
```

Note that the (sample) IP address given in the **[client]** section of the above **pc2v9.ini** file for 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 should use to contact the primary server and "join the contest".

A server which is started as a primary server in a multi-site contest waits for inbound connections from other servers; hence the primary server must allow such inbound connections (meaning, there cannot be any firewall blocking inbound connections at the specified port on the primary server). In a contest with *more than two sites* (hence, two or more secondary servers), each secondary server will also normally attempt to contact the other secondary servers. This means that secondary servers must likewise not have any firewall blocking inbound connections at the specified port. However, this restriction can be overcome (that is, secondary servers can be operated with firewalls blocking inbound connections) by using a special attribute called **proxyme** in the **pc2v9.ini** file. See the Appendix on *Networking Constraints* for details on how to use this attribute to allow firewall blocking on secondary server machines.

4.2 Other Initialization Files

For backwards compatibility, PC^2 still supports two additional initialization files: **reject.ini** and **sites.ini**. The **reject.ini** file was used in older versions of the system to specify the "reject messages" which judges could send to teams when a submission failed to solve a problem, while the **sites.ini** file was used to specify information for additional sites in a multi-site contest. Both of these functions have been incorporated into the configuration mechanisms described in the chapter on *Interactive Contest Configuration*, and use of these additional initialization files is deprecated and support for them may be removed in future versions of the system.

See Appendix P for more information about the format and content of the reject.ini.

5 PC² Startup Procedures

5.1 Built-in Commands

Once PC² has been installed and the necessary "`.ini`" files have been properly set up (i.e., edited and placed in the appropriate startup directory), the normal PC² 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 PC² 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.[10] To invoke the specified function, simply type the corresponding command at a command prompt. The commands reside in the "**bin**" directory beneath "$PC2HOME" (the root directory of an unzipped PC² installation), so the normal method of invoking them would be to change to the contest directory (that is, the directory where the contest will be run from, where logs are to be kept, etc.), and type the command name (note that this assumes, as previously recommended, that the $PC2HOME/bin directory has been added to the $PATH).

Command	Function
`pc2server`	Starts a PC² Server
`pc2admin`	Starts a PC² Client expecting an Administrator login
`pc2team`	Starts a PC² Client expecting a Team login
`pc2judge`	Starts a PC² Client expecting a Judge login
`pc2aj`	Starts a PC² Judging Client ("AutoJudge") in headless (non-GUI) mode
`pc2board`	Starts a PC² Client expecting a Scoreboard login
`pc2submit`	Submits a run using a command-line interface
`pc2extract`	Extract various information from a PC² server
`pc2ef`	Starts a PC² "Event Feed/Web Services" client
`pc2nc`	Starts a PC² "netcat" client for analytic/debugging purposes
`pc2report`	Generates various reports without needing the PC² Server running
`pc2reset`	Resets a contest, deleting all runs and (optionally) other data
`pc2ver`	Displays the current PC² version
`pc2zip`	Creates a "zip" file containing all the PC² contest data

The normal startup procedure, for example, would be to invoke the command "**pc2server**" to start a server, then *in a separate command window* to invoke the command "**pc2admin**" to start an Administrative client to be used to configure the contest details in PC². Once the contest details are configured, other clients can be started (as well as servers at other sites) using the appropriate commands.

[10] 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.

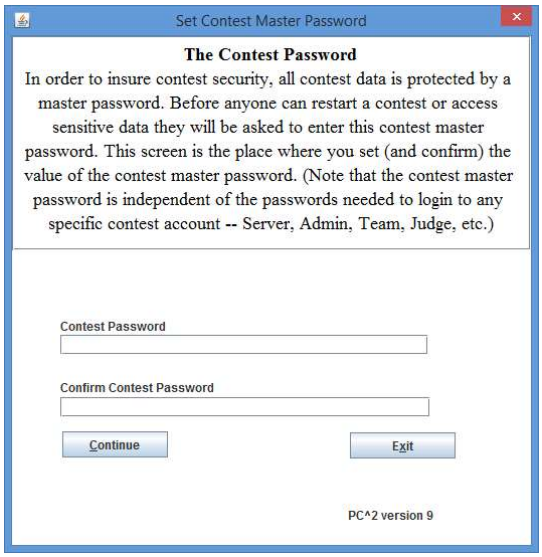
Additional details on the various PC^2 command scripts can be found at http://pc2.ecs.csus.edu/wiki/PC%C2%B2_scripts.

5.2 Server Startup

When a server is started (using the command "**`pc2server`**"), the user will see a login window similar to the following:

The default login name for the primary (first) site server is **site1**; the default password is also **site1** (the password can subsequently be changed; see below). Entering these values and pressing the Login button on the above screen will initiate the login process. However, there is a second step which must be performed to complete the login process.

At the time a server is started for Site 1 for the *very first time*, there is no contest-specific information stored in the system. All contest information which is subsequently entered will be stored in *encrypted* form, to protect the integrity of the contest data. In order to manage the encryption and allow authorized access at a later time, the Contest Administrator must provide a *contest master password.* Thus, on the very first login to a primary (first) server, the following screen will appear:



The Contest Administrator must enter (and confirm) a password to be used to perform contest data encryption. Note that *this password provides access to all contest-related data*; it should be well-chosen and well-protected. Note also that *there is no default value for this password;* it must be set (chosen and confirmed) by the Contest Administrator. Entering and confirming a contest master password (in addition to the login ID and login password) completes the login process.

If a first-site server is started again at some later point in time (i.e. after a contest master password has been set), pressing the Login button on the server's main login screen displays the following screen:

At this point the user must enter the Contest Password in order to complete the login process.

Only the primary (first) server in a contest prompts for the contest master password. In a multi-site contest the user starts a (secondary) server using the **`pc2server`** command as above, and logs in using the default site name (for example, **site2**) and password (**site2**). The secondary server contacts the primary server (pointed to by the **remoteServer=xxx** entry in its **`pc2v9.ini`** file) and obtains the current contest master password directly from the remote server. Note that the second server does *not* display the obtained contest master password; if people at secondary sites need this password for some reason then they must obtain it from the Contest Administrator at the primary (first) site.

5.2.1 Non-GUI Server Startup

In some environments it may be desirable to run a PC^2 server without a GUI front-end. This can be accomplished using the **`--nogui`** command line argument described in the appendix on Server Command Line arguments. Note that if this option is chosen there is no GUI-based way to enter login, account password and contest password information. These may be entered on the command line as well. For example, to start a primary (first) server with no GUI the following command could be used: [11]

```
pc2server --nogui --login site1 --password site1 --contestpassword contest
```

To start a secondary server without a GUI in a multi-site contest the following command could be used: [11]

```
pc2server --nogui --login site2 --password site2
```

Note that since this server is presumed to be secondary (meaning it has a **`pc2v9.ini`** entry pointing to a remote server which is the primary server), the **`--contestpassword`** option is omitted. This is because secondary servers obtain the contest master password directly from the primary server, as described above.

When a server is started using the **`--nogui`** option, it sends all of its output in text form to the console window from which it was started. The following shows an example of a server started in this way. Any subsequent text produced by the server (e.g. error messages or informational text) would appear on subsequent lines on the console.

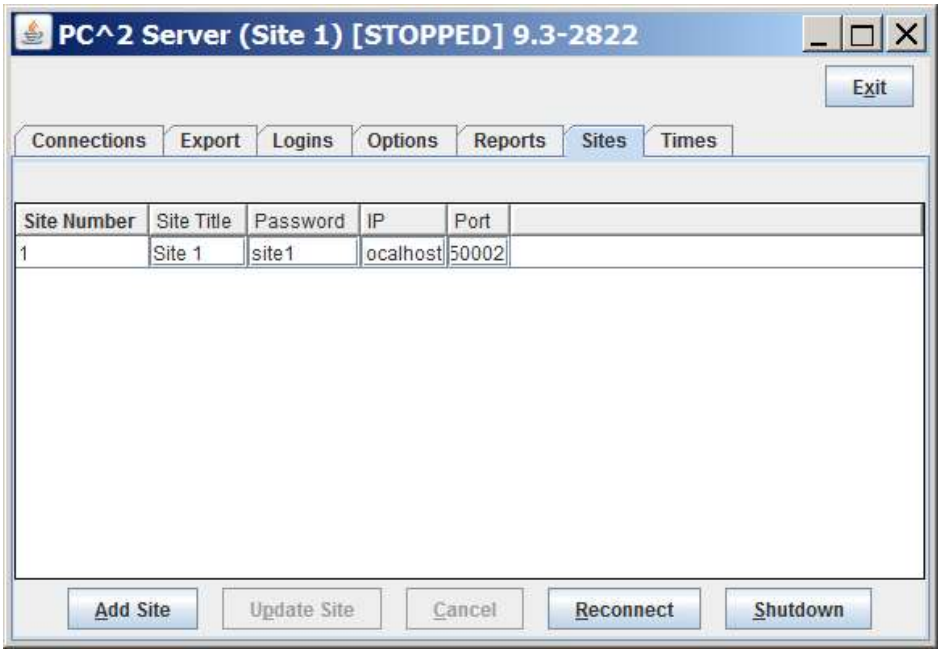
```
CSUS Programming Contest System
Version 9.3 20140802 (Saturday, August 2nd 2014 20:46 UTC) Build 2822
Java ver 1.7.0_55
Windows 7 6.1 (x86)

Date: 8/12/14 3:11 PM
Working directory is C:\pc2-9.3.0

Tue Aug 12 15:11:46 PDT 2014 Using Profile: Default @ profiles\P3a9add10-5089-47c5-
b82a-31966b136b6b

Tue Aug 12 15:11:46 PDT 2014 server (Site 1 - Site 1) started
```

To halt a non-GUI server, use the Shutdown button on the Site tab on the Administrator GUI to gracefully stop the server.

5.3 Server GUI Controls

Upon successful login to a server (assuming the server was started without the **`--nogui`** option) a GUI similar to the following appears:

[11] Note that a drawback of this approach is that the login and contest master passwords must be typed in plain-text on the command line. A much more secure option is to take advantage of the "-F" command line argument, which allows specification on the command line of a *file* containing the necessary security information. See the appendix titled "PC^2 Server Command Line Arguments" for details.



The tabs across the top of this GUI allow the user to examine and modify various configuration items in the contest. Those items which are primarily server-related are described below; some of the tabs also appear in the PC^2 Administrator GUI and are described in the chapter on "*Interactive Contest Configuration*". [12]

5.3.1 Adding Sites

The **Sites** pane (shown above) lists each contest site which the system knows about. Initially only "Site 1" is known; in order for a server at another site to join the contest, the additional site must first be added to the Sites list.

To make the system aware of the presence of another site, press the **Add Site** button to create a new row in the grid.[13] In the new row which appears, select the **Password** field and *change the password from the default value.*[14] Next, select the **IP** field in the new row and enter the IP address for the new site (that is, the IP address of that site's server machine); then select the **Port** field in the row and enter the port number at which the new site is expected to be

[12] In the case of starting a server with the "`--nogui`" option the PC^2 Administrator client is the *only* way to access some of these screens. In addition, some server-specific capabilities can *only* be accessed via the server GUI.

[13] The Add Site button also appears on the **Sites** tab in the Administration module GUI (where it can be found under the **Run Contest** tab).

[14] It is *critically important* for the security of your contest that you enter *new passwords* for *every* site (including the first one). Otherwise, since the default values for site passwords are well-known (published for example in this manual), some bad-guy could *start his own server and connect to your contest while it is running.*

contacting the primary site.[15] Optionally, select the **Site Title** field and assign a name to the site. Finally, press the **Update Site** button to save the site info.

Note that adding a site is a two step process: use Add Site to input the site data then Update Site to save the site data.

When starting a server for a site, the user must supply two data values: a login name and a password. For each site in the contest, the server's login name is the word "***site"*** followed immediately (no spaces) by the *site number* (for example, ***site1*** or ***site3***). Each server's *site number* is the value given in the leftmost field in the **Sites** display pane. The *password* for logging in to each site server is the value given in the **Password** field. If the password for a remote site is changed using the **Sites** pane, the new password must be relayed to the remote site in order for them to be able to log in to the contest.

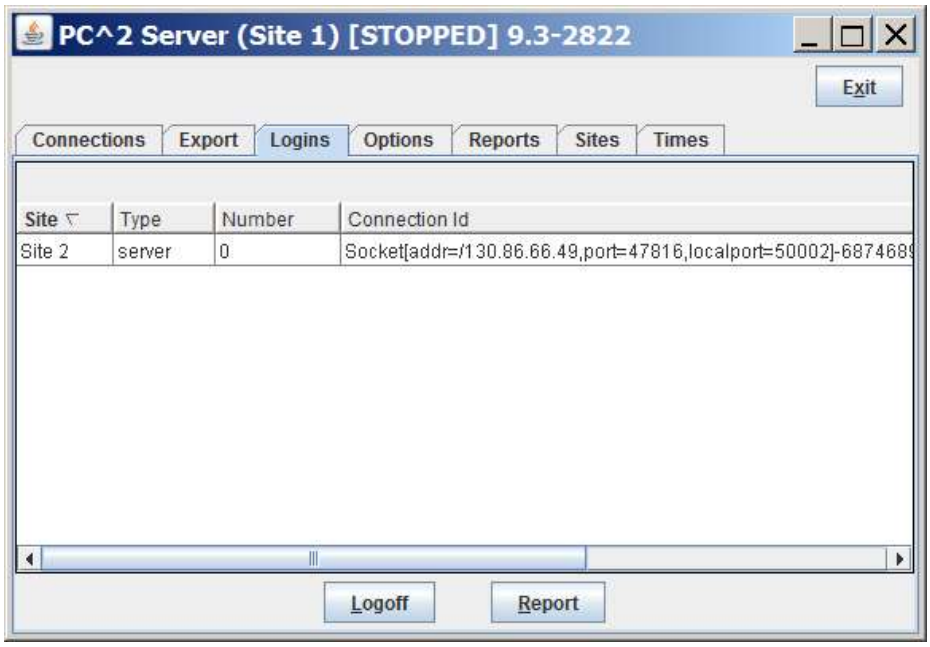
Note that there is no correlation between the value shown in the **Site Title** field of the **Sites** pane and the data required to log a server into a contest; login names for servers are always "***siteX***", where '***X***' is the *site number* shown at the left of each row in the **Sites** pane. The only function of the "Site Title" field is to provide a convenient human-readable reference for each site; that reference string is not used in any internal operations in PC^2.

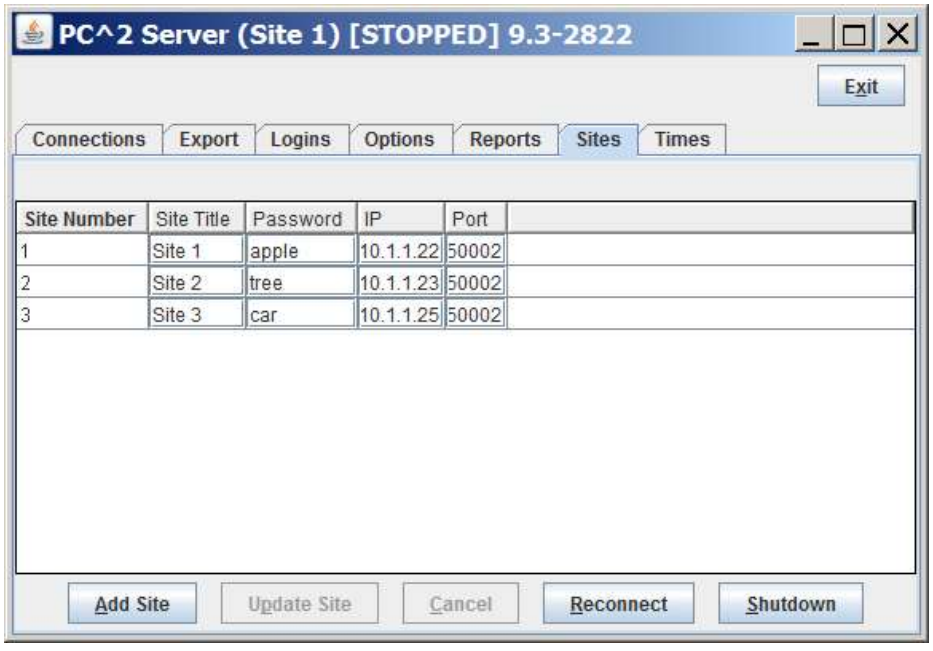
5.3.2 Restarting / Reconnecting Servers

During a contest there may be a loss of connectivity between sites, or a situation where one or more servers get "out of sync" with the others. In PC^2 Version 8 one had to shutdown (kill) the server and restart it when this happened. In Version 9 there is a *Reconnection* feature available to help with this situation.

To determine whether a site to which communication has been lost can be reconnected without killing and restarting the site's server, check the **Logins** pane (see below). If the server from that site appears in a row in the **Logins** grid, there is a connection between the sites. In this example on Site 1, Site 2 is logged in.

[15] Care should be taken to set the port number to the port which will be specified in the **`pc2v9.ini`** file at the remote site; the servers will not be able to communicate with each other if the port numbers do not match. Also note that when a new "site row" is added to the grid using the **Add Site** button, the default port number assigned to the new site is *not* the same as the default for the first site. Typically the port numbers must be changed to match (assuming all sites will use the same port for communication).





To reconnect a site that was previously logged in[16] but is now disconnected, use the Reconnect button. **Note that the Reconnect button is on the Sites tab, not the Login tab**. When the site is reconnected, that site will appear in the list on the Login tab.

[16] To reconnect a site the site must have previously been started. Reconnection will not restart a remote server.

5.3.3 Connections and Logins

These two tabs on the server GUI show what network connections and what PC^2 client logins have occurred since the server was started. They can sometimes be used to help with reconnection when sites get disconnected, as described above, or to trace anomalous or error conditions in the system. They can also be used to force a disconnection or logout from the system; this is done by highlighting (selecting) a row in the corresponding grid and pushing the **Disconnect** (or **Logoff**) button.

5.3.4 Additional Server GUI Controls

The remaining tabs on the Server GUI (**Export**, **Options**, **Reports**, and **Times**) are replicated on the Administrator GUI and are described in the chapter on *Interactive Contest Configuration*.

5.4 Starting Clients

Once a PC^2 server is running at a site, users (Contest Administrators, Judges, and Teams) at the site can start PC^2 clients to login and use the system[17]. The normal procedure is first to start a client using the "`pc2admin`" command and login as the "root" administrator (password "administrator1") 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[18]. The Contest Administrator would normally also start a PC^2 scoreboard using the "`pc2board`" command, logging in using the PC^2 account "board1".

Each time a client is started, the client will read its `pc2v9.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 `pc2v9.ini` file, the progress of these steps will be displayed and/or written to a file in the client's startup directory under **logs**. If any errors occur or the client fails to produce the login screen, check the log file in the **logs** directory for more details. [19]

[17] 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^2 graphical output from the client back to the Xterminal.

[18] With PC^2 Version 9.3 and above there is a web-based team client which allows using a browser on team machines instead of installing and running the PC^2 Team Client. See the *Team Clients* appendix for further information.

[19] See the Appendices for further descriptions of log files and their contents.

Note: PC^2 has a security rule that disallows multiple simultaneous logins for an account. For example, if a team logs into their account, and then tries to login again (say, from a different machine), the first login session will be terminated. This rule applies to all forms of login; for example, if a team is logged in via the Team Client GUI and later uses the **pc2submit** script to login and submit a run, the script login will cause the GUI login to be terminated. The same is true of using the external web-based team client; logging in via a browser will terminate any other current login for that team.

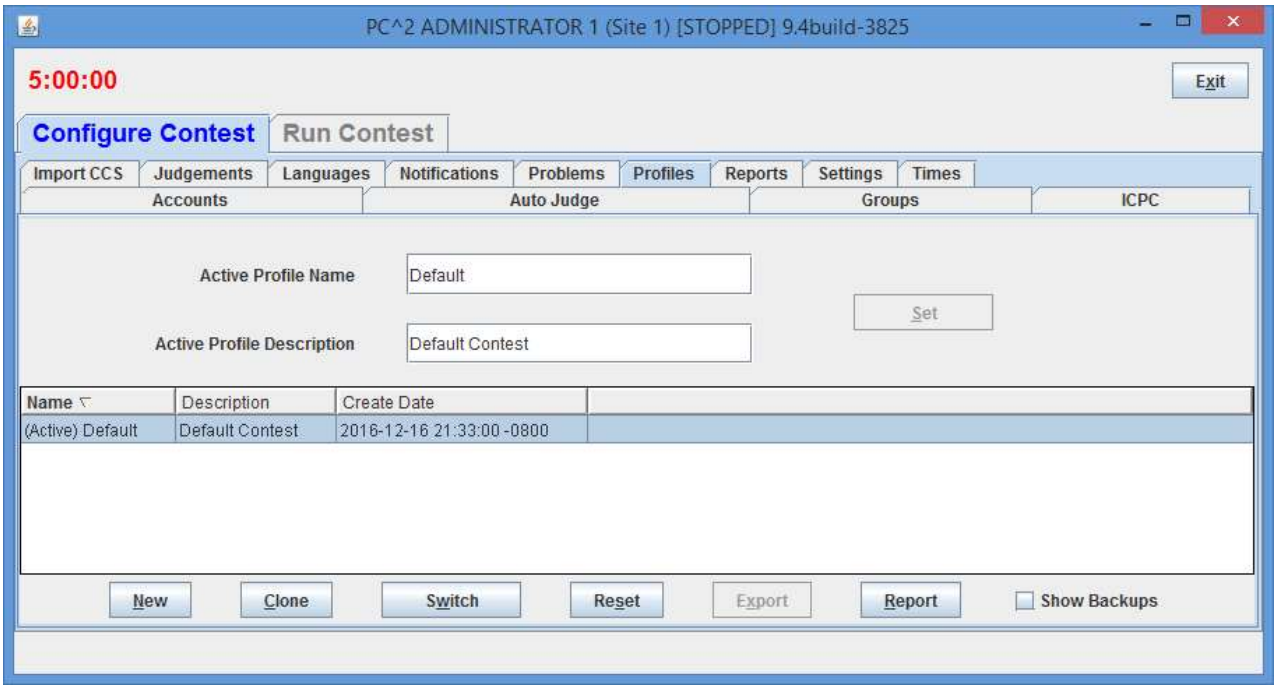
5.5 Contest Profiles

PC^2 provides a facility called *contest profiles.* A *profile* is a collection of all the information necessary to run a contest in a particular configuration (the accounts, languages, problems, scoring settings, etc.) Profiles allow the Contest Administrator to configure a contest, save the contest profile, and then configure a different contest. Switching between contest configurations is then just a matter of switching profiles.

This facility is useful, for example, when it is desired to run a Practice Contest followed immediately by a Real Contest using the same Languages and Accounts but with a different problem set. The Contest Administrator would configure the Practice Contest, save the current profile under the name "Practice", then "clone" the configuration into a differently-named profile (for example, "Real").

Options allow selectively cloning only certain configuration data; for example, you can suppress copying the problems, runs, and clarifications from the Practice profile into the Real profile, while resetting the contest time in the Real profile and keeping the same Languages, Accounts, Groups, Judgment Names, Notifications, and so forth. Switching from the Practice Contest to the Real Contest is then simply a matter of switching between profiles.

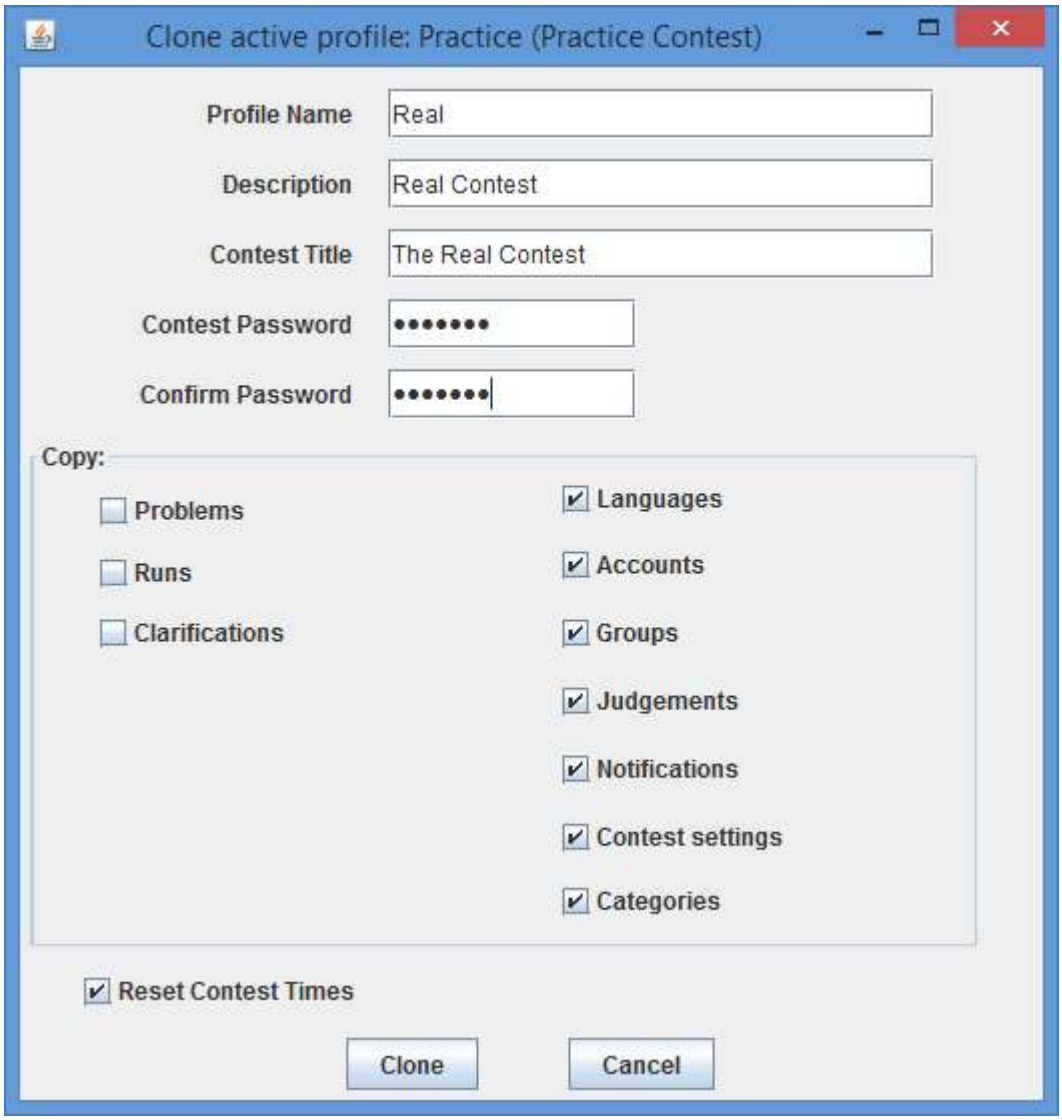
Profiles are created using the **Profiles** tab on the **Configure Contest** screen of the Admin client, shown below. NOTE: **Profiles are only supported in a *single site* contest**.[20]



Pressing the **NEW** button on the **Profiles** screen displays the following dialog which allows creation of a new profile. Note that new profiles are essentially empty – meaning that the user is responsible for providing all configuration information for the profile, including some information that PC^2 normally automatically provides as part of the "Default Profile". To avoid having to enter all configuration information for a profile, use the "Clone Profile" operation (see below).

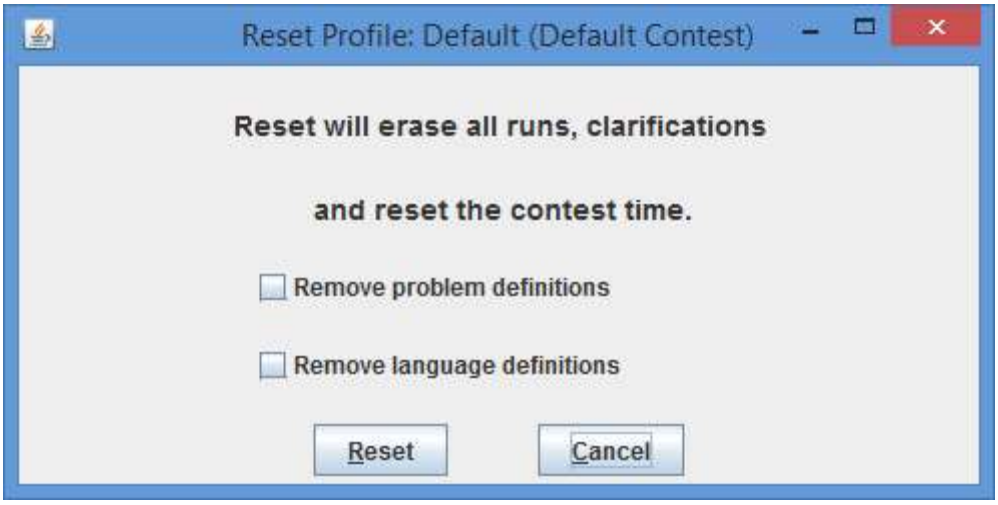
[20] "Single site contest" refers to having only a single PC^2 Server running; it is perfectly possible to use Profiles in a contest with teams at geographically separate locations, by having all teams connect to the same PC^2 Server either using the Team Application Client or the web-based team client.

Pressing the **CLONE** button on the **Profiles** screen displays the following dialog which allows creating a clone (copy) of the *currently active* profile. Notice that in the example below a currently active profile named "Practice" is being cloned to produce a new profile named "Real"; all the information which is the same between the "practice" and "real" contests (for example, Languages, Accounts, Judgements, etc.) is being copied over to the new (clone) profile, but the Contest Problems, Submitted Runs, and Submitted Clarification Requests from the Practice Contest are *not* being copied. This provides a simple mechanism for duplicating the configuration between practice and real contests without having to reenter all the configuration information.



The **SWITCH** button on the **Profiles** screen is used to switch the system between profiles. To perform a switch, the new profile (the one to which the user wishes to switch) must first be selected by clicking on its row in the table of profiles. Pressing the **SWITCH** button then pops up a dialog as shown below, asking for the "master contest password" associated with the selected profile (the one being switched to). Entering the password and then pressing the **SWITCH** button on the dialog causes PC^2 to switch its configuration to the configuration defined by the selected profile. The contest must be stopped for the Switch operation to be allowed.

The **RESET** button on the **Profiles** screen is used to reset various configuration elements in the *currently active* profile. (Note that even if another profile is "selected" by clicking on its row in the table, **RESET** applies to the currently *active* profile; to reset a profile other than the active profile you must first use **SWITCH** to activate that other profile.) The contest must be stopped for the Reset operation to be allowed. Pressing **RESET** displays the following dialog:



Pressing **RESET** on the dialog will remove all submitted runs and clarifications from the currently active configuration, as well as resetting the contest time. Checking the **Remove**

Problem Definitions and/or the **Remove Language Definitions** checkboxes will cause the corresponding configuration data to also be removed when **RESET** is pressed.

When a profile is "reset", PC^2 automatically makes a backup copy of the profile as it appeared before it was reset. This allows the user to recover from unintended actions caused by a Reset; it is always possible to switch back to a previous version of a profile by switching to its backup. Note however that backup profiles are not by default displayed in the table of profiles. Checking the **Show Backups** checkbox on the **Profiles** screen causes ALL profiles (including backups created by a Reset) to be displayed.

Pressing the **REPORT** button will generate a PC^2 Report of the known profiles (including backup profiles). See the Appendix on PC^2 Reports for additional information.

6 Interactive Contest Configuration

Configuring a contest involves at least the following: creating user accounts for teams and judges; defining contest languages; and defining contest problems. It can also include a variety of additional steps such as configuring judgement messages, balloon notification handling, site configuration (for a multi-site contest), and setting various options such as scoring properties, output size limits, and information visible to judges.

Contest configuration can be done in two ways: *interactively* via various Administrator screens, or by *loading configuration data from files*. The following sections describe the various interactive configuration steps. See the separate chapter, **Configuring the Contest via Configuration Files**, for information on loading configuration data from files rather than interactive configuration.

Note: the chapter on configuration via files is somewhat terse, in that it does not describe all of the nuances of various configuration items – only how to specify the configuration items via configuration files. If you encounter topics in that chapter which are unclear, refer back to the corresponding sections in this chapter for additional details.

6.1 Administrator Login

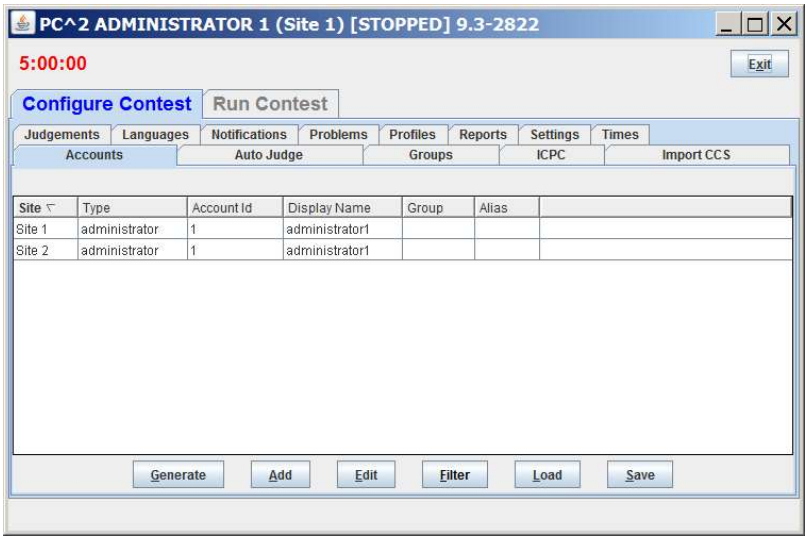
Once PC^2 is set up and running, it is necessary to "log in" to the system via a client login window in order to use the system. A PC^2 "account" is required in order to log in. Initially only a *single* account exists; the account name ("login ID") of this single account is "**root**". This account is a *master Administrative account* which is used to configure the PC^2 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 **administrator1**. 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 account management functions on the "Configure Contest Accounts" tab; see below.

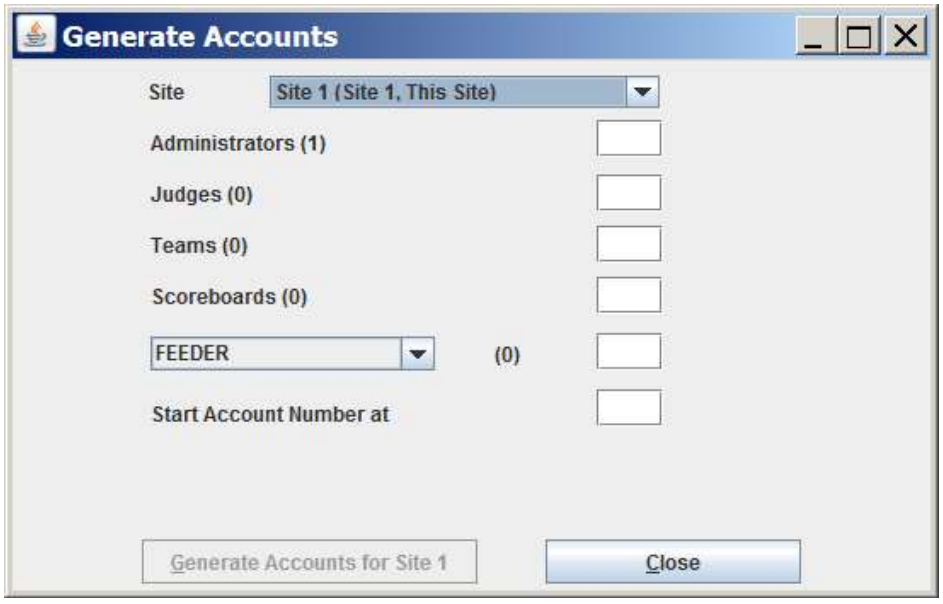
After logging in to the "**root"** Administrator account, the following screen will be displayed. This is referred to as the "main Administrator screen". It provides a series of "tabs" across the top to select various contest administration functions. The tabs which are used to configure the contest 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 Configure Tab on the main Administrator screen. This will display the following screen:



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 (or automated judge) who will be judging the contest at this site, and at least one Scoreboard ("board") account if the PC^2 scoreboard is going to be used 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^2 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.

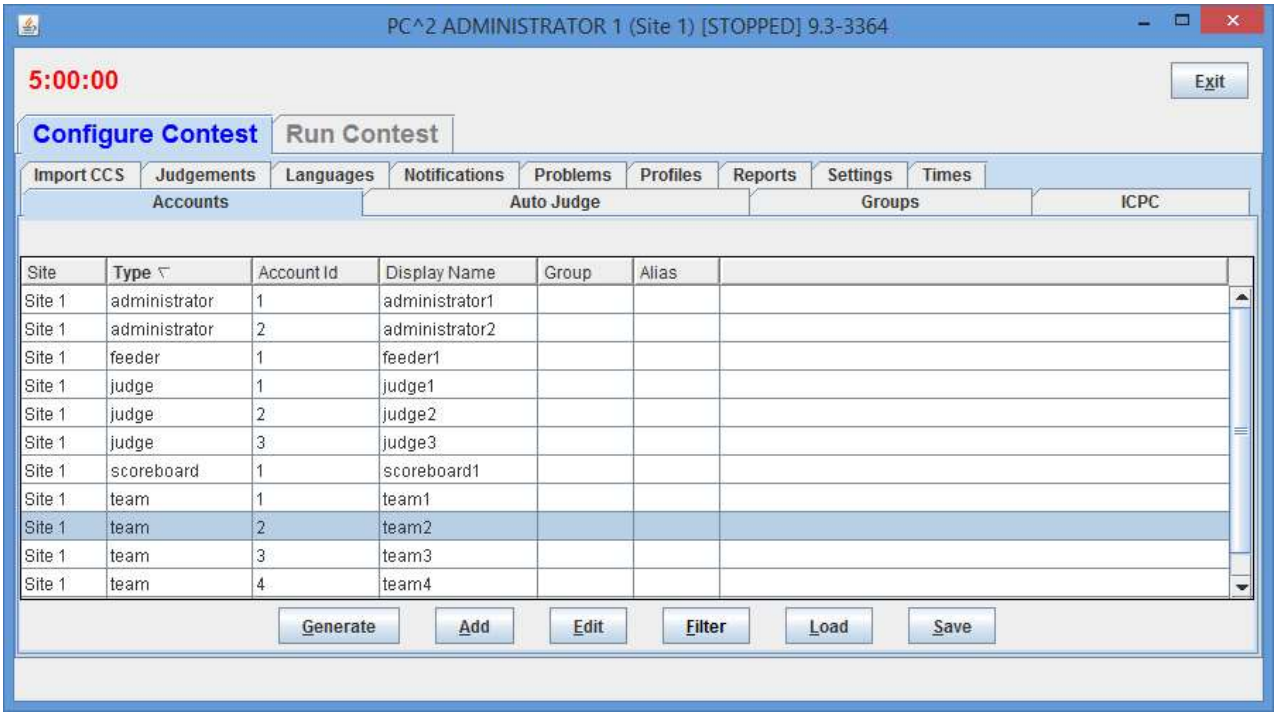
Accounts in PC^2 are *site-specific*. This means that in a multi-site contest an Administrator must create the accounts for each site. There are two steps for the Administrator to create accounts on a site. First, start a PC^2 server at each of the sites. Second, an Administrator creates accounts by selecting the site then creating the accounts as described above.[21]

Note: by default PC^2 only allows a *single* login at a time for any account; if a second user logs into the same account using the same (valid) credentials, the first login session is automatically terminated (this is a security feature to insure that teams are not using more than one computer to access the PC^2 system). However, starting with Version 9.7, PC^2 provides the ability for the Contest Administrator to configure the system to *allow* multiple logins *for team accounts*. See the section on **Options (Settings)**, below, for details on allowing multiple team logins.

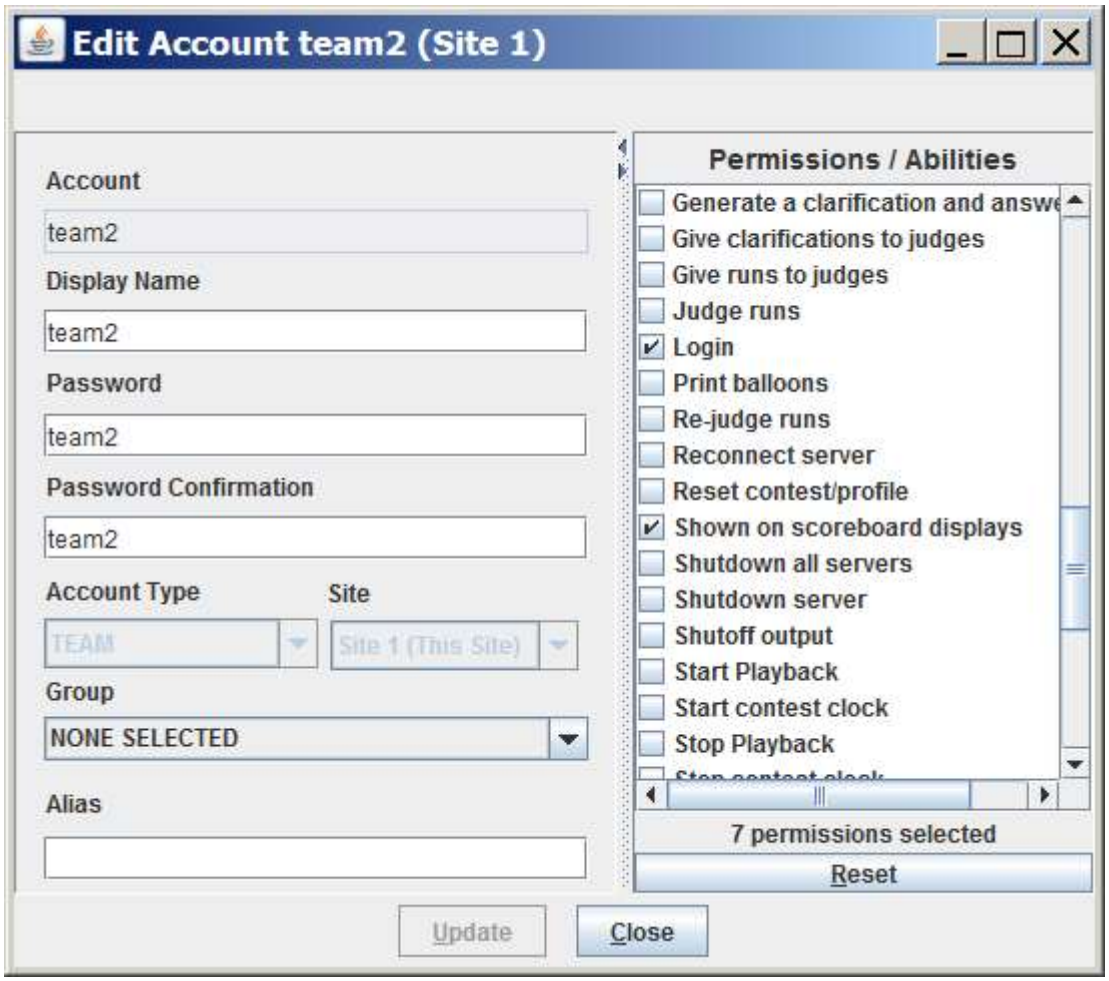
6.2.2 Account Names and Passwords

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 is created with all letters in lowercase.

[21] Recall that, as mentioned earlier, it is actually not usually necessary to use "multi-site" mode – that is, to run more than one PC^2 server. Assuming reliable network communications are available, teams and judges at arbitrary physical locations can normally connect to a single PC^2 Server running the entire contest.



Passwords for accounts can be changed from their default values by editing each account. 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^2 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 Permission "Shown on scoreboard displays" 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 "Shown on scoreboard displays" checkboxes – otherwise they will appear on the scoreboard as teams which have solved no problems. *Note that the "*Shown on scoreboard displays*" checkbox only determines whether a team appears on the scoreboard*; teams which are not shown on the scoreboard 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 or uncheck the "Login" Permission checkbox)

The "Group" field is used to associate accounts with different "regions" or "groups". This is used in conjunction with the PC^2 scoreboard for displaying rankings of different subgroups (see the chapter on the PC^2 Scoreboard for further details).

Note that PC^2 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^2 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^2. This can be done by preparing an "account data" file and using the **Load** button on the Accounts tab load the data into the system.

An "account data load file" consists of a series of text lines: a single line that defines the account data fields that will be loaded, followed by lines which contain information for each account.

The format of the account data load file is as follows. File lines starting with ! or # in the first column are treated as comments and are ignored. Each non-comment line has fields, separated by a <tab> (ASCII 9).

The first line of the file must contain the names of the fields to be loaded, separated by tabs. Subsequent lines contain field data, one line per account. The site and account fields are required; other fields are optional. The fields may appear in any order (except that the field order on data lines must match the order specified in the first (header) line). The recognized field names are:

- **`site`** – site number
- **`account`** – team login name (ex. team1, judge4, scoreboard2)
- **`password`** – account password
- **`group`** – group name
- **`displayname`** – name to be displayed on scoreboard
- **`alias`** – an alias display name shown to judges to preserve team anonymity
- **`permdisplay`** – true or false, display on scoreboard
- **`permlogin`** – true or false, allowed to login

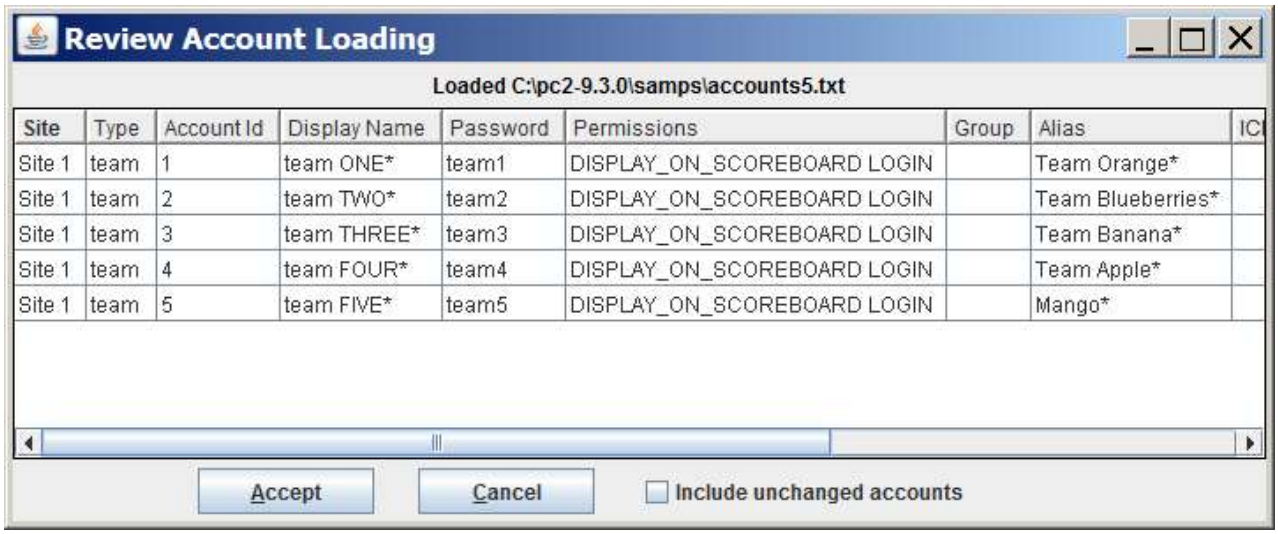
For example, to initialize accounts "team1", "team2", and "team3" so that the "team1" account displays on the scoreboard with the name "Number 1" and has a password of "pass1", while the "team2" account displays on the scoreboard with the name "Team Number 2" and has a password of "myPass", and the "team3"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

```
site<tab>account<tab>displayname<tab>password<tab>permdisplay
1<tab>team1<tab>Number 1<tab>pass1<tab>true
1<tab>team2<tab>Team Number 2<tab>myPass<tab>true
1<tab>team3<tab>My School Name<tab><tab>false
```

Note: for clarity in the example above the tab character is represented as `<tab>` ; it should appear as a single tab character (ASCII 9) in the actual file.

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 site and account fields determine which accounts will be modified (any unlisted accounts will remain unchanged).

To load the account load file use the Load button on the Accounts Tab (under the Configure tab). The load button will display a File Open dialog; select the name of the account load file and click Ok. At this point the Review Account Loading dialog will appear, as shown below:



Any changes/differences that will be applied will have an asterisk at the end. Click on Accept to apply the changes. By default only accounts that have changes are shown, check the "Include unchanged accounts" checkbox to see all accounts that were loaded.

6.2.4 Importing ICPC Data

PC^2 was designed for supporting the 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.[22] PC^2 provides interfaces to import

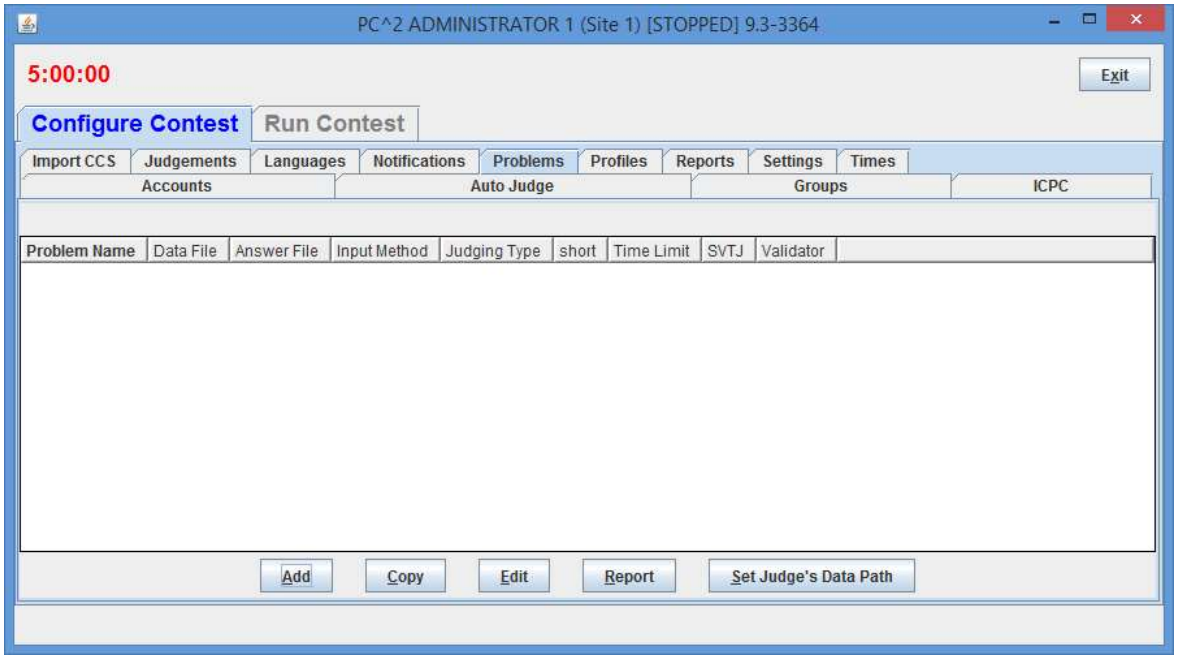
[22] Visit the ICPC web site at http://icpc.baylor.edu/icpc/ for further details.

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.

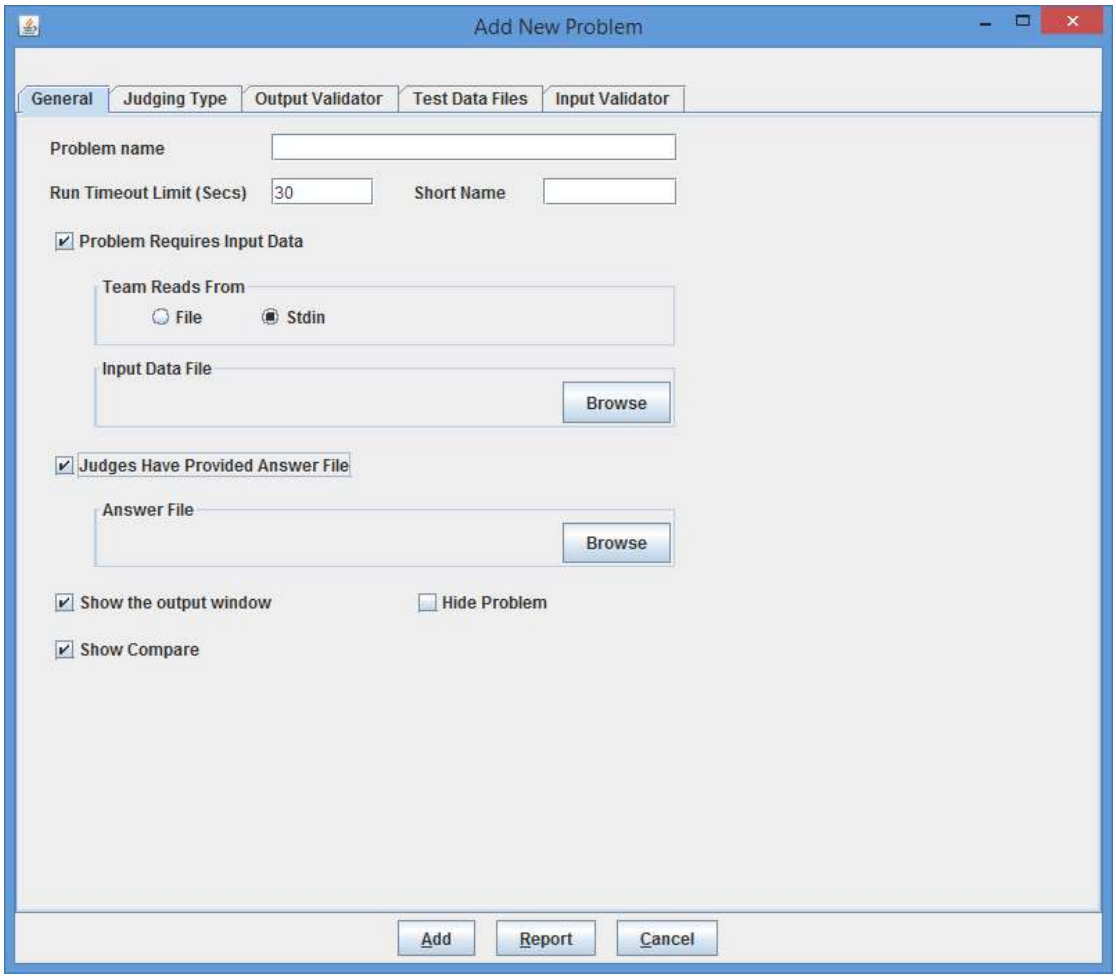
6.3 Contest Problems

6.3.1 Defining a Problem

PC^2 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 Configure Contest 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 "**Add New Problem**" dialog shown below.



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

1) Enter the problem name in the top textbox.

2) Enter the value (in seconds) that the system should enforce as the problem run time limit.

3) Enter a "short name" for the problem (this is the unique string by which the problem is known internally in the system).

4) If the problem requires an input data set, click the "Problem Requires Input Data" checkbox and then

a. 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[23], then

b. if the problem is to be tested using only a single data file, use the Browse button to select the data file (see below for how to define a problem which is to be tested using multiple judge's input data files).

5) If the Judges have provided a single "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. (See below for how to define a problem with multiple Judge's Answer Files.)

6) Select the desired "Display Options" to be applied when the problem is executed and displayed on the Judge client:
   - Show the output window – shows the PC^2 output window upon completion of execution/validations
   - Show Compare – shows the PC^2 compare window upon completion of validation
   - Hide Problem – do not show this problem to the teams (also suppresses showing this problem on the Scoreboard HTML/output)

7) Click the **Add** button to save the problem description.

As each contest problem 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 defining a contest problem:

- The Run Timeout Limit value (shown as 30 in the sample screen above but settable to any positive integer number of seconds when the problem is defined) is enforced by PC^2. A count-up timer is displayed during program execution so that the Judge can tell how long the program has been executing; when the specified timeout limit is reached the program will be terminated and the Validator judgment will show "No - Time Limit Exceeded". The timer also includes a button to allow the Judge to terminate the program at any time.

- The content of the input data file for a problem is stored internally when the **Add** button is pressed (that is, PC^2 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 press the **Update** button. Upon pressing the Update button, a prompt will appear

[23] 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^2 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).

confirming that the file has changed on disk. Answer Yes to the prompt to re-load the data file.

- PC^2 automatically associates a "problem letter" with each defined contest problem; the first problem entered becomes "Problem A", the second problem entered becomes "Problem B", etc. Since Version 9.7 problems after the 26th defined problem ("Z") are lettered "AA', then "AB", etc. The only place where problem 'letters' are actually displayed is on the PC^2 scoreboard (which displays the problem letter in lieu of the full problem name).

- All team program output is expected to go to "standard output" (where it is captured by PC^2 and saved for display to the Judges). More specifically, there is no mechanism in the current version of PC^2 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).[24]

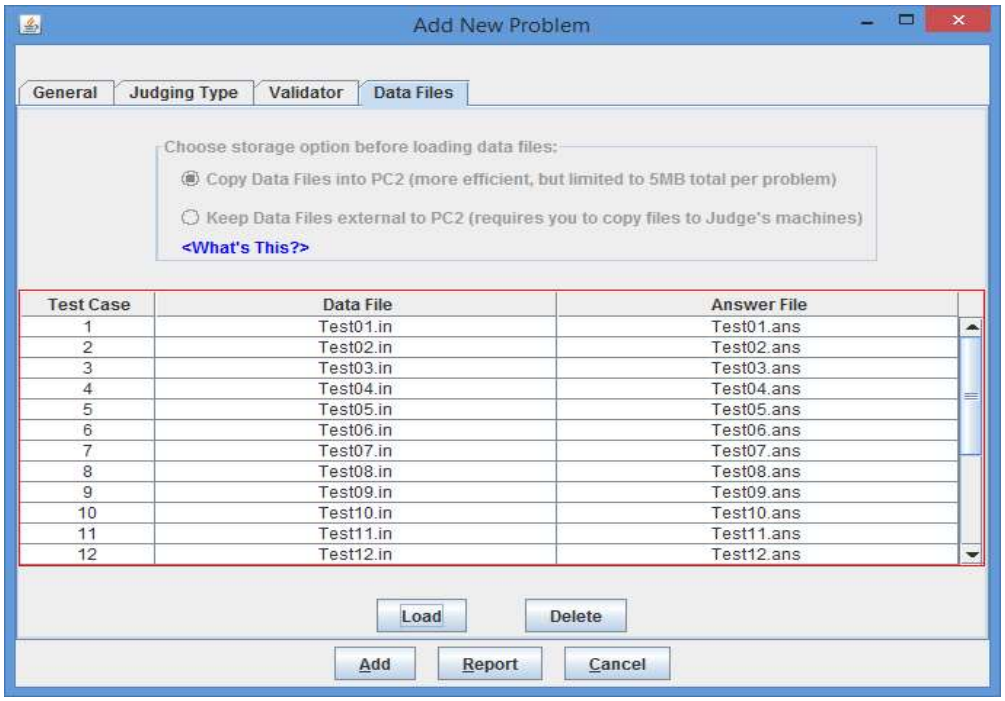
- Contest problems in PC^2 are *global*, in the sense that once a problem definition is entered by the Contest Administrator that problem definition is broadcast to all sites. There is no mechanism for having teams at different sites in a multi-site contest see different descriptions of the same problem. 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^2 views a contest as a set of teams all working on an identical problem set. Note also that contest problems appear in the same *order* at each site in a multi-site contest.

- Care must be taken when configuring contest problems in one machine environment (say, Windows) and then moving the configured system to a different environment (say, Linux). In particular, problem files (judge's data files and judge's answer files) should not be configured this way. PC^2 records the *path* to configured files; the path will most certainly be different when the configuration is moved to a different platform.

[24] There is in principle no reason a contest administrator could not use the PC^2 "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.

6.3.2 Multiple Test Data Files

The previous section described how to configure a Contest Problem which is to be tested using a single data set (or "test case"). PC^2 also supports the ability to automatically execute a single team submission against multiple different data files. In order to use this capability, the input data files and corresponding answer files[25] must be loaded using the **Add Problem** or **Edit Problem** screen's **Test Data Files** tab, shown below (shown with a set of input data and answer files already loaded):



The "storage option" radio buttons on the **Test Data Files** tab are used to control the way in which PC^2 handles storage for data files. By default, data files are automatically loaded into PC^2 system memory (this is what "Copy Data Files into PC2" means). Subsequently, whenever a human judge or PC^2 "Auto-Judge" (see below) is used to judge a problem, the data files are automatically shipped to the judge along with the submission. This works fine for most contest problems; however, extremely large data files (e.g. larger than around 5MB) can cause system memory overflow and/or cause heavy network traffic loads.

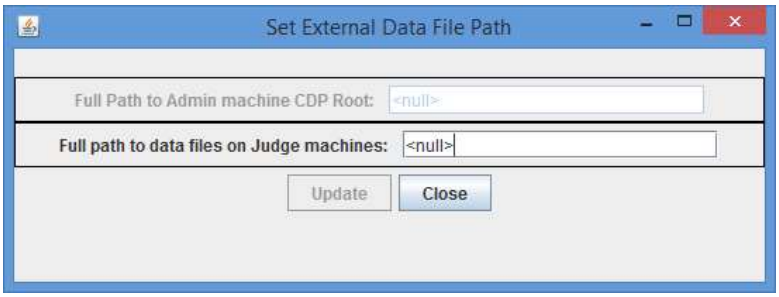
To help mitigate these problems, data files can be marked as being stored external to the PC^2 system ("Keep Data Files External"). In this configuration, PC^2 records necessary

[25] When using multiple test data files, each data file *must* have a corresponding "answer file".

information about the data files (e.g. the file names), but does not actually load the data into the system and does not transmit the data to judges. This eliminates the memory and network traffic issues. However, *it requires the contest administrator to insure that all the data files for the problem are copied onto all Judge's machines.*

Note that *the storage option for a problem must be set* ***prior to loading the data files for the problem***, and *cannot be changed once a problem is created and saved in the system.*

When using external data files it is also necessary to configure PC^2 to be aware of where the data files are located when they are stored on the judging machines (by the contest administrator, external to PC^2).[26] Specifying the external data file location is done using the "**Set Judge's Data Path**" button on the main Administrator screen's **Problems** tab (see the earlier screen), which pops up a dialog for setting the path as shown below.



The Judge's Data Path *must* be set whenever a contest problem is configured to use external data files (the system will not allow to save a contest problem configured to use external data files without setting the Judge's Data Path). Further, when a PC^2 Judge is started it will automatically check to verify the required data files are present in the specified location on the Judge's machine and will refuse to perform judging if the files are not found in the specified location.

Once the data storage option for the problem data files is selected, press the **Load** button on the **Test Data Files** tab to load the set of test input data and answer files into the system. This will pop up a window that allows you to navigate to the place where the data and answer files are stored. Using the navigator, select the *folder which contains the set of input data and answer files.* The result of loading a set of input data and corresponding answer files is show in the display above.

PC^2 follows the CLICS CCS standard which requires that input data files have names ending with the extension ".in" and that answer files have names ending with the extension ".ans". Selecting a folder causes the system to load all data files and answer files matching those names from the selected folder into the system. If no such files are found, or if there is a mismatch between the number of data and answer files, the system displays an error message.

Loading a set of input data and answer files means that when a problem is judged, it will be executed once for each input/answer file pair, and the results of each of these executions will

[26] Setting the Judge's Data Path is necessary because the data files might be stored under a different path on the Judge's machines from where they are located on the Administrator's machine – which is allowable (as long as the files are located in the *same* path on *all* the Judge's machines).

be displayed for the judge (see the separate PC^2 Judge's Guide for details on the display of execution results to the judge).

During judging, PC^2 executes multiple test cases by processing the input data files in *lexicographic order of file names.* This means for example that a set of three input data files with the name *"test1.in", "test2.in",* and *"test10.in"* would be executed in the order "test1.in", then "test10.in", then "test2.in". To avoid confusion in naming of input test data files, be sure to specify the names in the desired lexicographic order. (For example, put leading zeroes (e.g. "001", "002", "010") in all numeric file name elements to keep "1" and "2" from being separated by "10".)

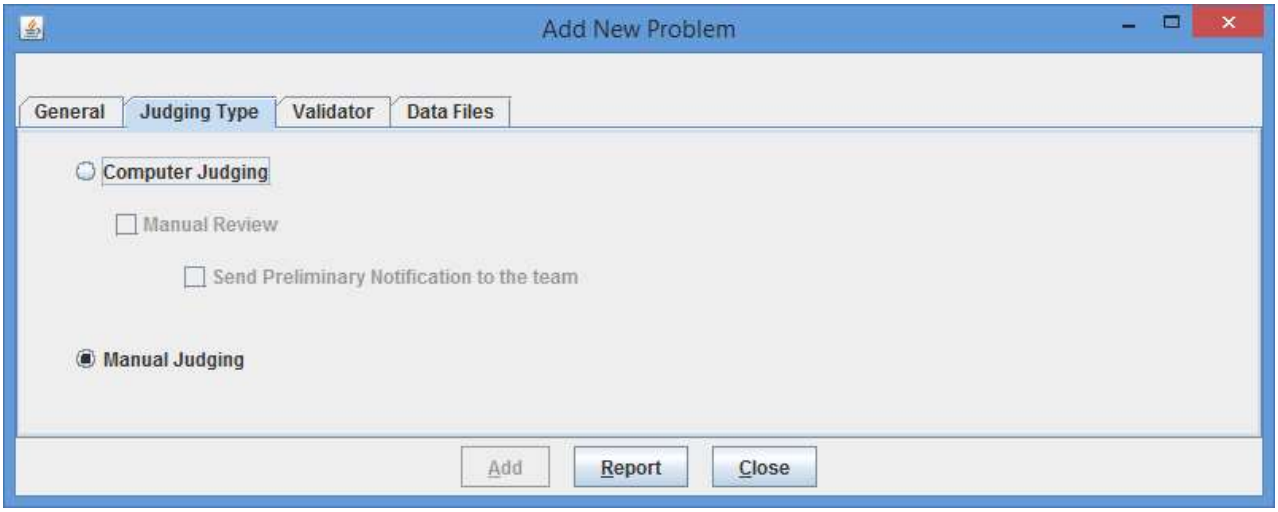
6.3.3 Defining Judging Type

By default each contest problem is manually judged, meaning that a person selects a judgment for each submitted run. The system can also automatically judge runs, and when doing so can also include an optional second step where a human judge manually reviews and judges the run.

Configuring the system to automatically judge a problem requires several steps: (1) a *validator* must be defined for that problem[27]; (2) the *Judging Type* for the problem must be set as *Computer Judging*; and (3) at least one judge module must be configured to automatically judge the problem (that is to say, there must be at least one PC2 "AutoJudge" running; see below). Note that these steps must be performed for *each problem* that is to be computer judged, although the same judge module can be used to do automated judging for more than one problem (see the following section for information on configuring judge modules for automated judging).

[27] A *validator* is a program that examines the output of a team's program and determines whether it is a correct solution to the problem. The **Validator** tab on the **Add Problem** or **Edit Problem** dialog is used to specify the validator program to be used for the problem. See the Appendix on Validators for further details.

Specifying that a problem is to be computer judged is done via the **Judging Type** tab on the **Add New Problem** dialog, shown below.



To specify computer judging (also called "automated judging"), perform the following steps:

1) Select **Computer Judging.**

2) Optionally select **Manual Review** if the problem is also to be judged (reviewed) by a human judge after the automated judging process is completed.

3) Optionally select **Send Preliminary Notification to team.**[28] The effect of selecting this option is that a notification of the preliminary "automated judging" result will be sent to the team. These preliminary notifications are clearly marked "Preliminary", and the judgment value they assign can subsequently be overridden by the human judge during Manual Review (see the separate PC2 Judge's Guide for more information on overriding preliminary judgments).

If the Judging Type for a problem is Computer Judging but a validator is not defined for that problem, the system will not allow the problem definition to be saved; instead a message "Computer Judging selected, must select a validator" will be displayed when an attempt to save the problem is made. Once a validator is defined then the problem definition can be saved. Again, refer to the appendices for information on defining validators for problems.

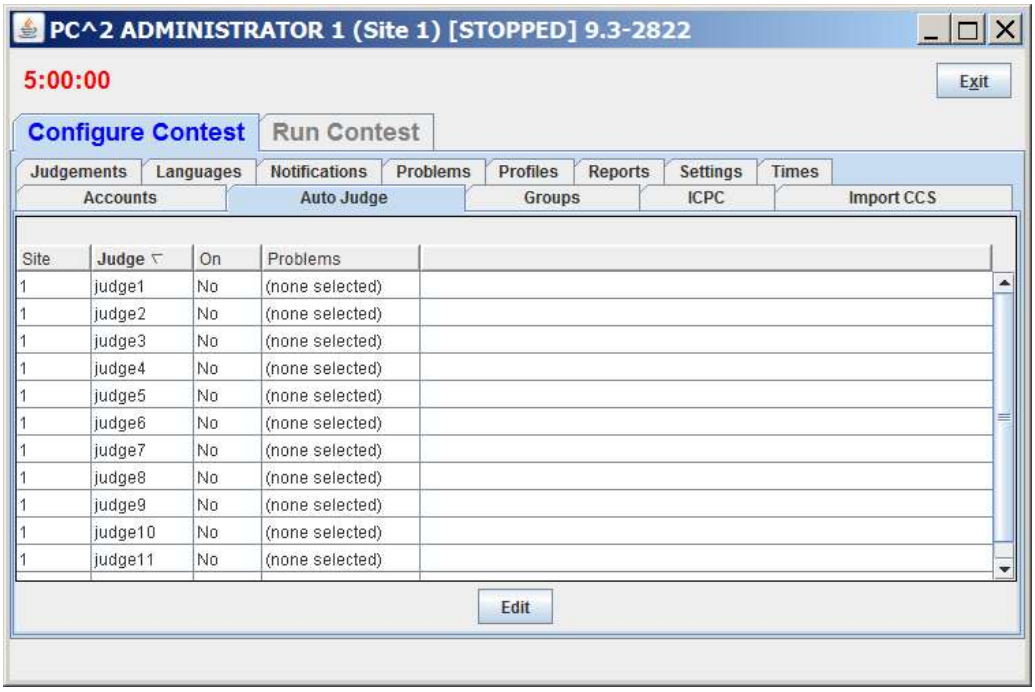
6.3.4 Assigning Auto Judging to Judge modules

In order to accomplish computer (automated) judging, each problem which is to be auto-judged must have at least one PC^2 judge module configured to be aware that the problem should be

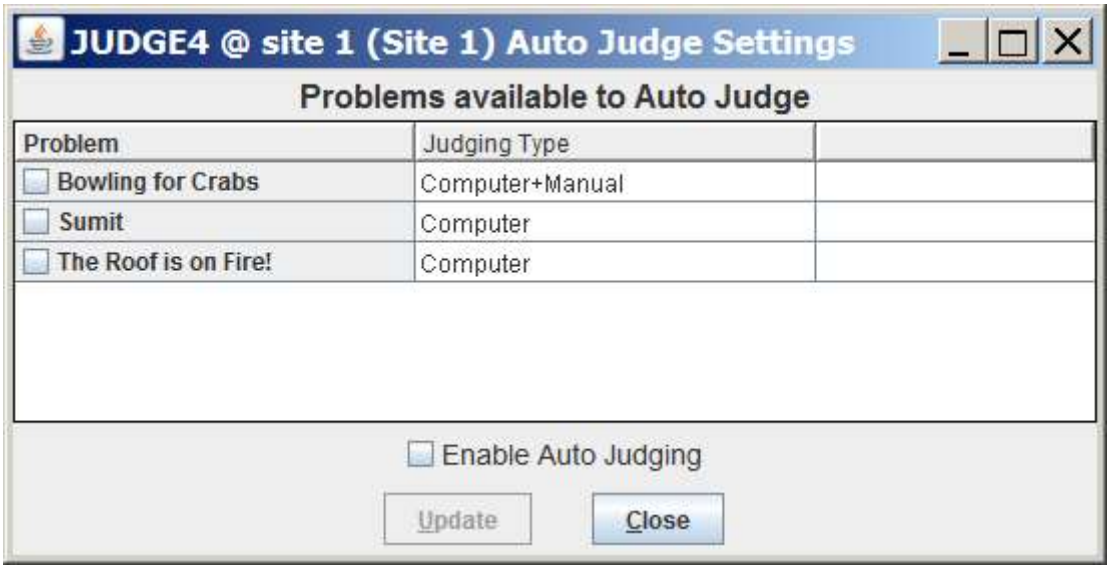
[28] If **Send Preliminary Notification to the team** is not selected when **Computer Judging** and **Manual Review** are selected, the automated computer judgment will not be shown to the team.

automatically judged. This can be accomplished in a variety of ways: a single judge module can be told to handle all auto-judge problems, or a separate judge module can be configured for each different problem, or the set of problems to be auto-judged can be distributed in some partitioned fashion across a set of PC^2 judge modules. The requirement is simply that, for each problem designated as requiring "Computer Judging" (as described above), there must be at least one PC^2 judge module made aware that that particular problem requires automated judging.

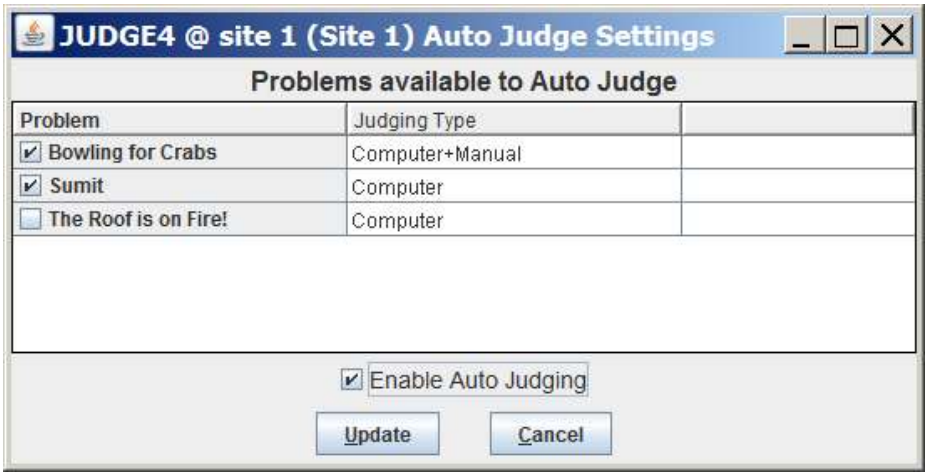
Configuring judge modules for auto-judging is accomplished by specifying, for each "judge login account", which problems (if any) that account should perform auto-judging on. (For this reason, auto-judging assignments cannot be made until after Judge login accounts have been created.) To configure judge modules for auto-judging, use the **Auto Judge** tab on the **Configure Contest** tab on the Administrator main screen, shown below.



To specify that a particular judge account is to auto-judge one or more problems, select the row identifying the judge account and click **Edit**. This will pop up the **Auto Judge Settings** dialog, shown below. The dialog will show (only) problems that can be automatically judged (that is, problems which have a validator defined and have been specified as requiring **Computer Judging**).

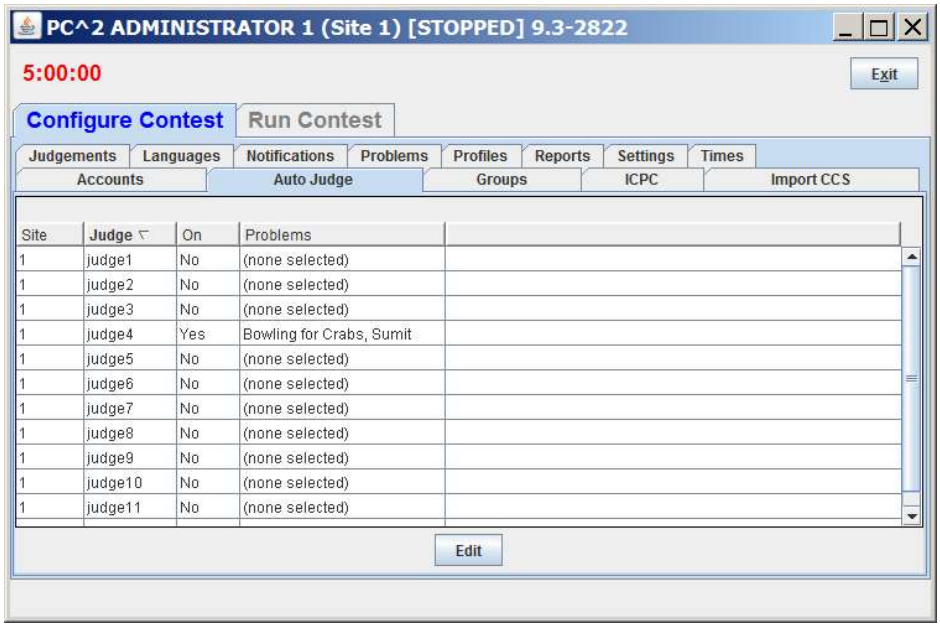


Select the problems that this judge module is to auto-judge by clicking on the corresponding problem row checkbox, then click **Enable Auto Judging** followed by **Update**. In the example shown below Judge 4 at Site 1 will automatically judge the problems named "Bowling for Crabs" and "Sumit", but will *not* judge the problem "The Roof is on Fire!".



Upon clicking **Update** the **Auto Judge** tab will change to show that auto-judging is ON for the specified problems for Judge 4 at Site 1, as shown below. From this point on, any time Judge 4 at Site 1 is logged in it will automatically fetch and judge runs for either of the two specified

problems (but will not judge any other problems). Runs will be automatically selected in chronological order and judged. To monitor the status of runs use the **Runs** tab under the **Run Contest** tab.



Any number of judge modules can be set up in this way, judging any combination of contest problems (provided that the problems have been configured with a validator and specified as "Computer Judging" problems).

Note that some consideration should be given to the assignment of problems to judge modules. For example, if one contest problem is known (or expected) to be likely to incur long run-times, it might be desirable to insure that more than one judge module is configured to auto-judge that problem. Likewise, it might be desirable to avoid assigning other problems to such a judge, since those problems will necessarily be delayed in their judging until the long run-time of the previously-judged problem has expired.

Note also that a judge module which is configured for auto-judging should not also be used for human judging (review); separate accounts should be used for human logins.

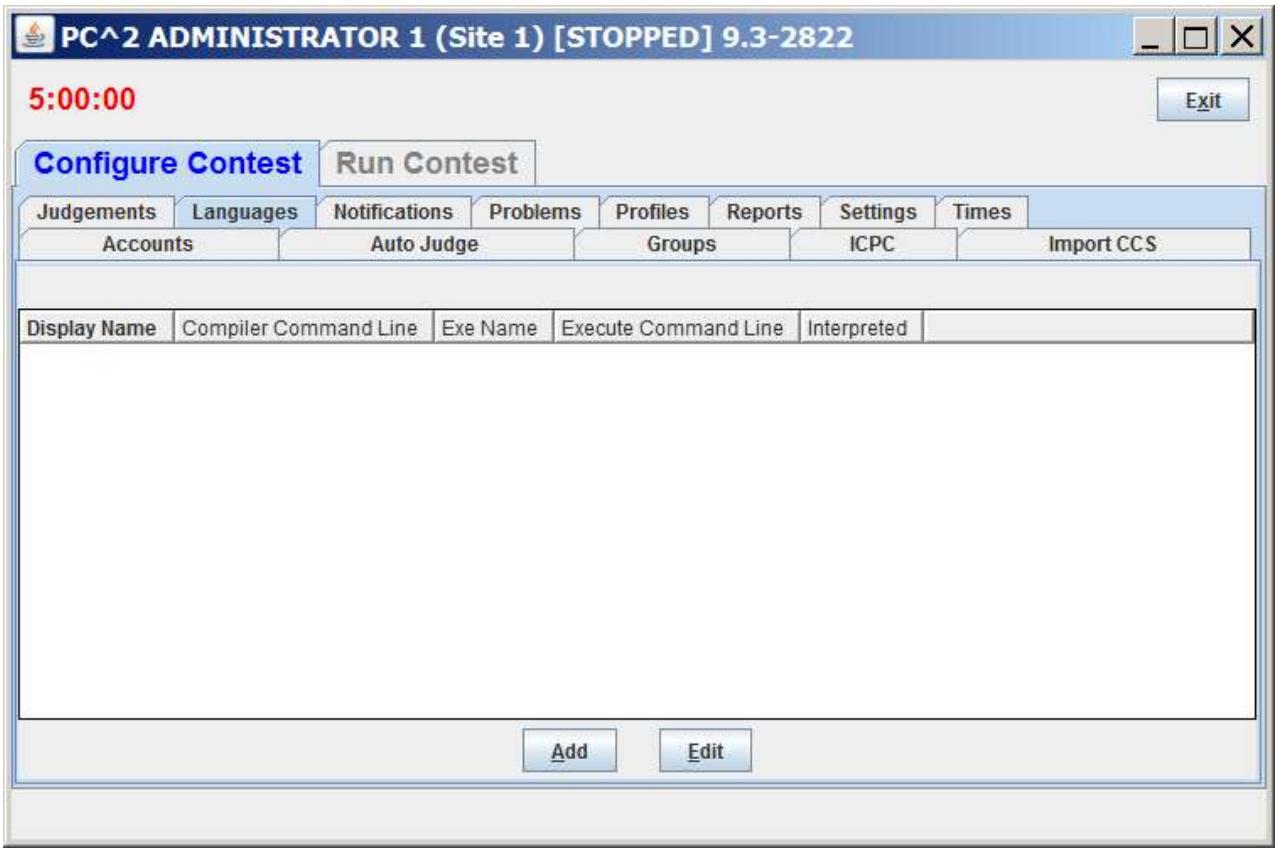
When a judge account is logged in then computer judging will start automatically on any problems for which that account has been configured for auto-judging; no additional steps are necessary to begin auto-judging.

To stop a judge module from auto-judging, Edit the Auto Judge (Settings) and uncheck the Enable Auto Judging Tab.

6.4 Contest Languages

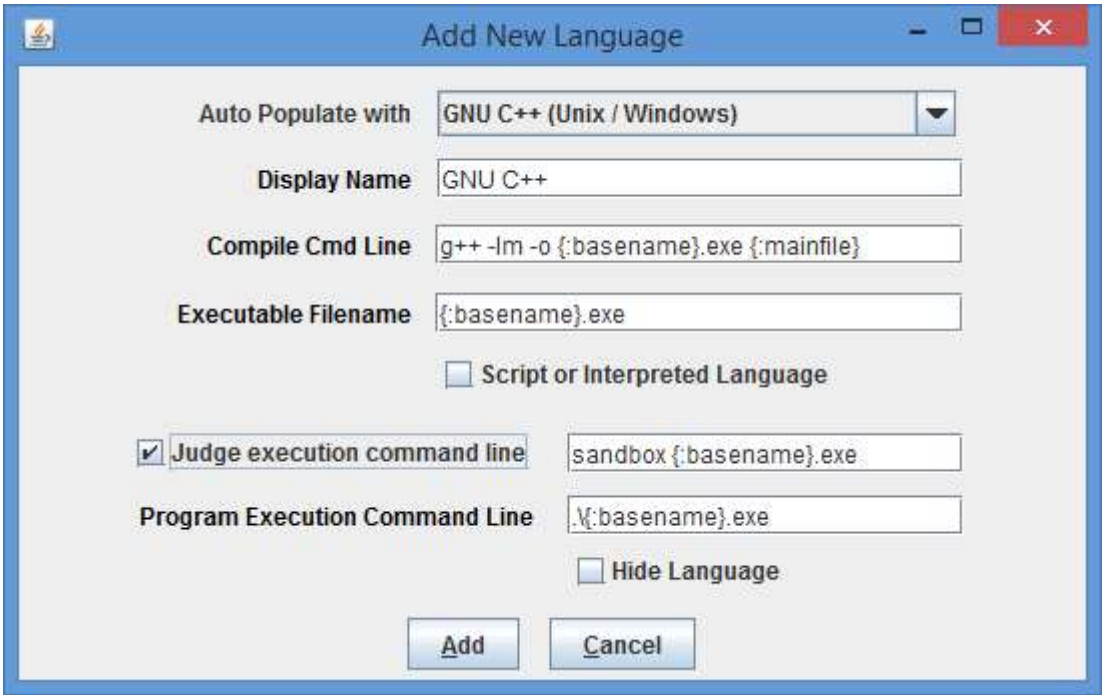
6.4.1 Defining a Language

PC^2 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 **Add New Language** dialog, similar to the one shown below, containing fields used to describe the language to PC^2.

PC^2 contains built-in descriptions for a number of commonly-used language compilers. These built-in descriptions can be selected using the "**Auto Populate**" drop-down list. In the example **Add New Language** dialog shown below, the Auto Populate function has been used to select the language configuration for Java. Note: the example also shows the use of something called "command parameter substitutions" – strings such as "**{:mainfile}**" containing curly braces. See the section on **Command Parameter Substitution** for an explanation of these strings.



The **Add New Language** fields have the following meanings:

- 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^2 the name (or more correctly, the form of the name) of the output (executable program) file produced by the compilation process. PC^2 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.

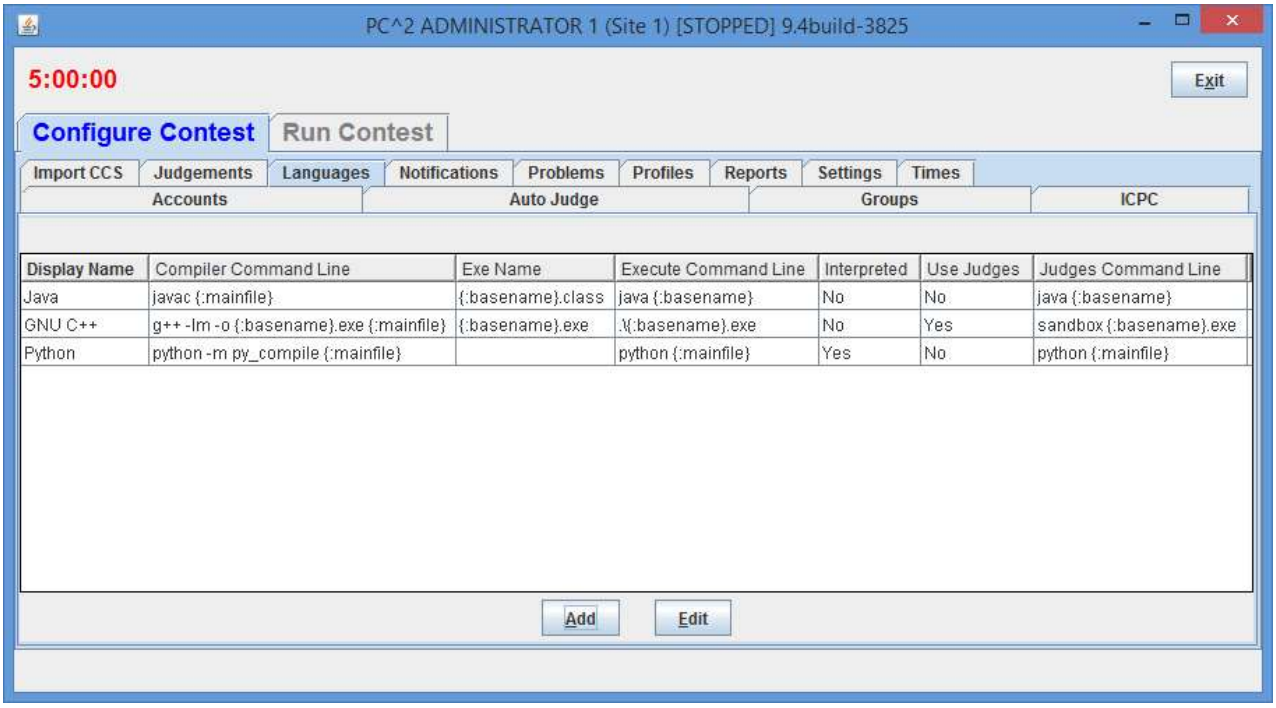
- The "**Judge Execution Command Line**" field is used to specify the (form of the) command line required to execute (run) the resulting program *on Judge machines*. It is only necessary to specify a value in this field if the form of the execution command is different on Judge machines than it is on Team machines. (This might be the case, for example, if the Judge machine is intended to run the program inside a "sandbox"; on a team machine the program would be directly executed but on a Judge machine the program executable would be passed to the sandbox for execution under controlled conditions.) The **Judge Execution Command**

Line field is ignored unless the adjacent Judge Execution Command Line checkbox is checked. If not, the Program Execution Command Line (above) is used to run the program. In any case, as with the Program Execution Command Line, execution using the Judge Execution Command Line is only performed if the preceding steps were successful in producing a new executable file.

- The "**Script or Interpreted Language**" checkbox should be selected (checked) if the language does not require compilation prior to execution (for example, Python, PERL, and Ruby are languages that do not compile first; the language source code is fed directly into an interpreter). Checking this checkbox tells PC^2 it should skip over any "compilation" step and go straight to the "execution" phase.

The Contest Administrator must define each language to be used in the contest by filling in the language definition fields (or populating them using the **Auto Populate** function). As previously noted, the example screen above 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 **Add** 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 **Add New Language** 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.



Note that care must be taken when configuring languages using the "Auto Populate" function. In particular, this function uses and records the current (platform-specific) path separators

(e.g. "\" under Windows vs. "/" under Linux). If a language is configured this way and then the configuration is moved to a machine of a different type, the strings which are used to invoke a compiler will be incorrect (they will contain the wrong path separators).

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^2 supports the use of "parameter substitutions" in these fields.

PC^2 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 a set of predefined PC^2 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.

Keyword	Meaning
`mainfile`	Replace with the full name of the submitted file, including any extension (but excluding any 'path' specifier on the front of the filename)
`basename`	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. For a complete list of keyword substitution variables support by PC^2, see the PC^2 Wiki at http://pc2.ecs.csus.edu/wiki/Variable_Substitutions.

6.4.3 Language Definition Examples

The language screen example shown above shows a set of filled-in fields defining a language named "GNU C++" and using the GNU g++ compiler.

The compile command line invokes the compiler ("**`g++`**") 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, with the characters "**.exe**" appended. (So for example if a team submitted a file named "**proga.cpp**", the object output file would be named "**proga.exe**", since that is the value to which the "**{:basename}.exe**" string would be expanded when parameter substitution is applied.)

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.cpp**" if that was the name of the submitted main program source file.

The Executable Filename field indicates that the executable file which is produced by the compile command has the same name as the base name of the submitted program, with ".exe" appended; 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 on a Team machine 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, with ".exe" appended.

The Judges Execution command field specifies that the command used to execute the compiled program on a Judge machine is the command "sandbox", which is passed a single argument: 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, with ".exe" appended.

If a team were to submit to the judges a C++ program in a file named **proga.cpp** using the above language, PC^2 would first execute:

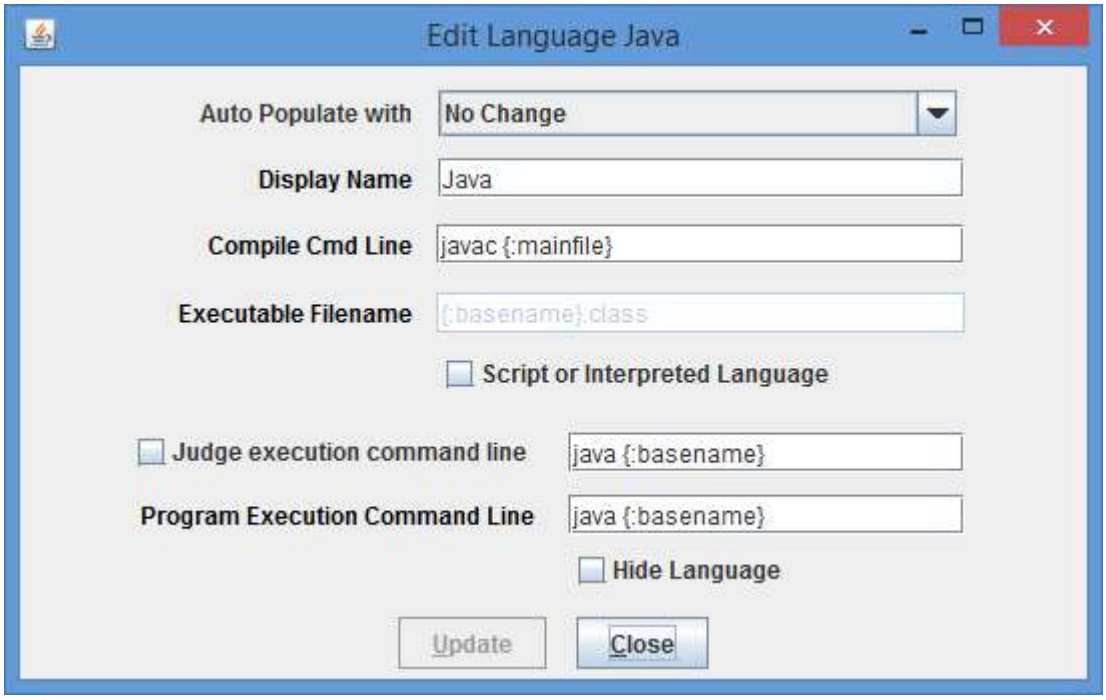
```
g++ -lm -o proga.exe proga.cpp
```

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

```
sandbox proga.exe
```

Note: in a Unix-like environment, the "**.exe**" appended to the output (executable) file name in the above example is not strictly necessary. However, in a Windows environment, executable files must in most cases have the extension ".exe". Explicitly adding the "**.exe**" to the language definition allows the same language definition to work in both environments.

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 to the judges a Java program in a file named **`sumit.java`** using this language definition, PC^2 would execute the following command to compile the program:

```
javac sumit.java
```

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

```
java sumit
```

Note that the form of the language definition fields differs somewhat between the first 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^2 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^2 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^2 using the actual language names ("C", "Java", and "Pascal") as the PC^2 "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^2 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 "*executePacal"* scripts for the program execution language definitions in PC^2 and then arrange for appropriately different script contents at each site.

Note also that PC^2 "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.

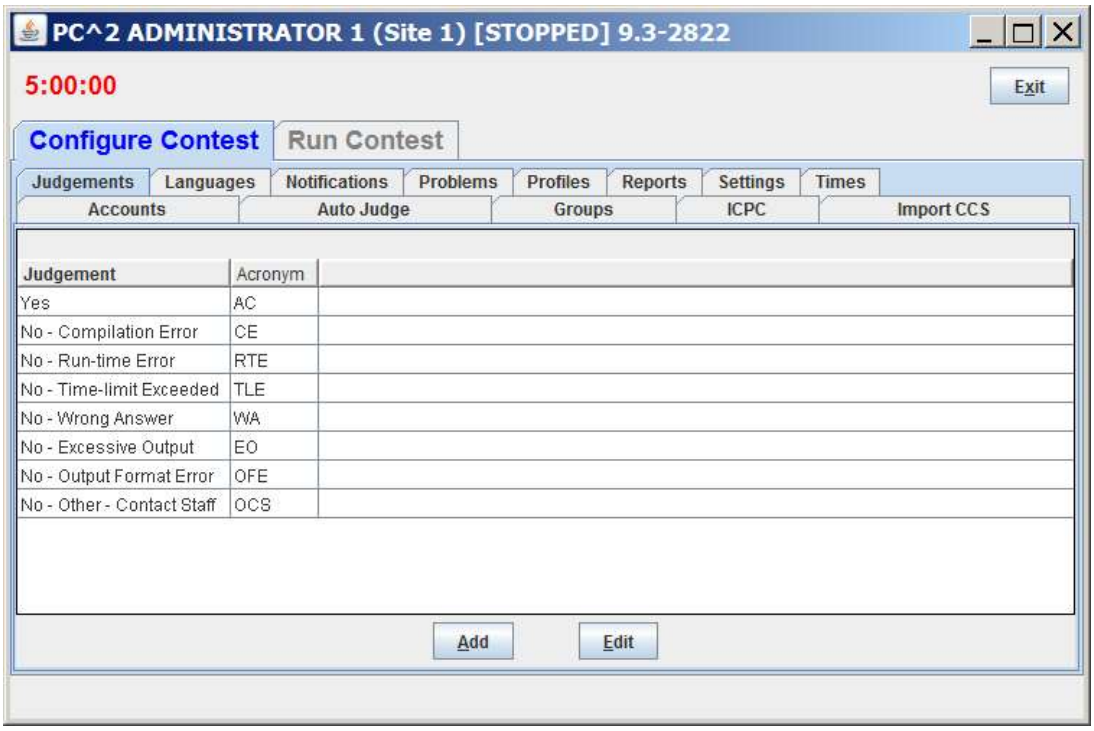
Using generic script names in PC^2 language definitions and providing site-specific implementations of each language script at each site allows the Contest Administrator to significantly reduce the number of language definitions which teams must deal with, while at the same time retaining the flexibility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

6.5 Contest Judgments

6.5.1 Defining a New Judgment

PC^2 uses the term "judgments" to refer to the possible responses which a judge (human or automated) can apply to a run submitted by a team. The Judgments Tab under the Configure Contest tab on the Administrator main screen shows all the judgments available. The set of judgment messages can be viewed, added, edited and deleted.

The following screen shows the set of judgments which are defined by default.



To add a judgment click the **Add** button. This will bring up an **Add New Judgement** dialog, similar to the one shown below.

The "Judgement" field defines the name which Judges will see when they are asked to judge a run. This name is also seen by the Teams when they receive that judgment.

The "Acronym" field defines the abbreviation used in the event feed and some reports.

The "Hide Judgment" checkbox will remove (hide) this judgment from the list of judgments that the Judges can use.

6.5.2 Changing Existing Judgments

As shown above, the set of default judgments in PC^2 is given by the following list:

```
Yes
No - Compilation Error
No - Run-time Error
No - Time-limit Exceeded
No - Wrong Answer
No - Excessive Output
No - Output Format Error
No - Other - Contact Staff
```

The contest administrator can use the **Edit** button on the Judgments tab to change the text of existing judgments. Selecting a judgment message from the list and then clicking the Edit button will bring up an **Edit Judgment** dialog, similar to the **Add New Judgment** dialog, allowing changes to be made to the judgment text.

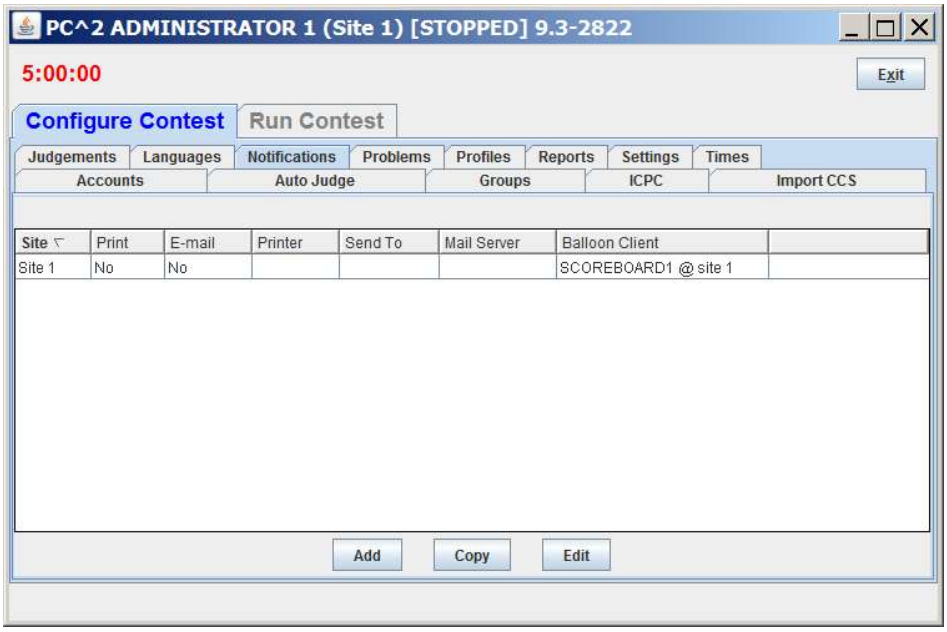
However, one guideline must be followed: PC^2 assumes that the *first* judgment in the list is always the "Yes" judgment – that is, the first judgment is the one which the system returns for problems judged to be correct. The text of the default first judgment message can be changed if desired (for example, the contest administrator may prefer the message "Accepted" instead of "Yes"), but regardless of the actual text in the first judgment field it is that text which will be returned for correct runs. Therefore it would be a bad idea to change the text of the first judgment message to some form of "No" or "Incorrect Run" message.

6.6 Balloon Notifications

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^2 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".[29] In particular, the system supports the creation of a separate "Balloon Notification configuration" for each site in the contest. Selecting the **Notifications** tab on the main Administrator screen will produce a display similar to the one below listing the currently-defined Balloon Notification configurations.

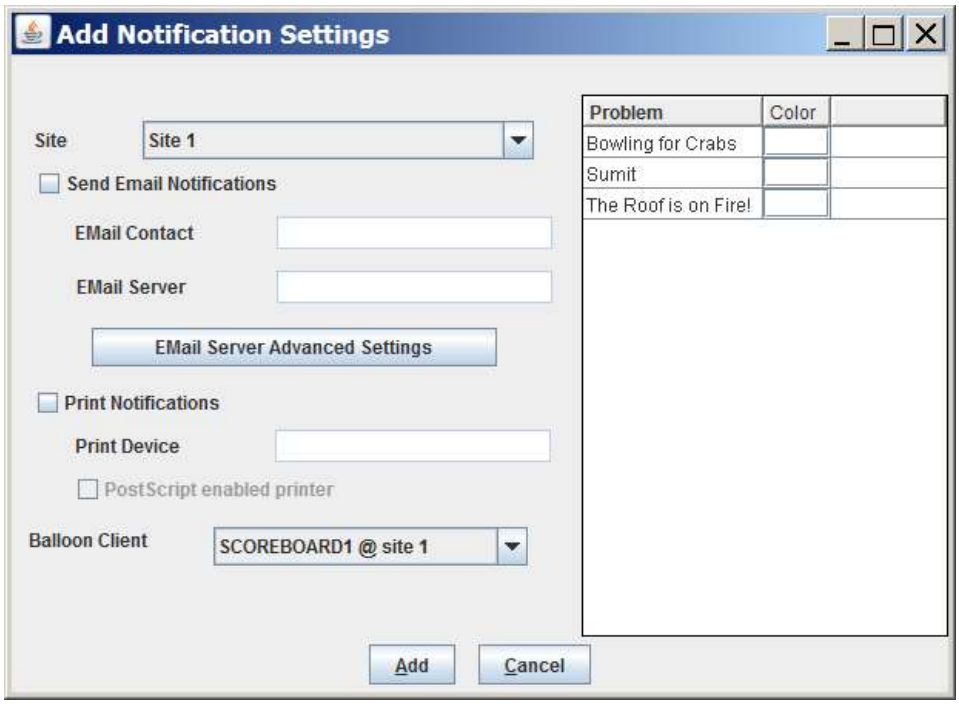
The sample screen below shows one Balloon Notification configuration already created (for Site 1); by default there are no such configurations and screen will be empty. Clicking the **Add** button will cause an **Add Notification Settings** dialog similar to the one shown in the next section to appear; that dialog is used to configure the handling of balloons for a given site. Selecting an existing row in the table and then clicking the **Edit** button will allow editing of the selected configuration. Selecting an existing row and then clicking the **Copy** button will create a copy of the selected settings and allow selection of a different site which should use those same settings (in a multi-site contest using the same settings this avoids having to reenter the settings).



[29] Note that while PC^2 has its own Balloon handling mechanism, it is also compatible with the "ICPC Balloon Utility" – the tool used at the ICPC World Finals as well as in a variety of Local and Regional contests around the world. The ICPC Balloon Utility is somewhat less complicated and more robust; we actually use that rather than the built-in PC^2 Balloon Notification system in most of our own contests. More information can be found at https://tools.icpc.global/.

6.6.1 Defining Balloon Notifications

Balloon Notification 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 is specified on a per-site basis (so that, for example, sites can use different color balloons for a given problem).



To enable the use of email balloon notifications for the specified site, check the "**Send Email Notifications**" box, then enter in the appropriate text boxes the full name of an SMTP email server accessible to the PC^2 Scoreboard machine along with a valid email address (EMail contact).

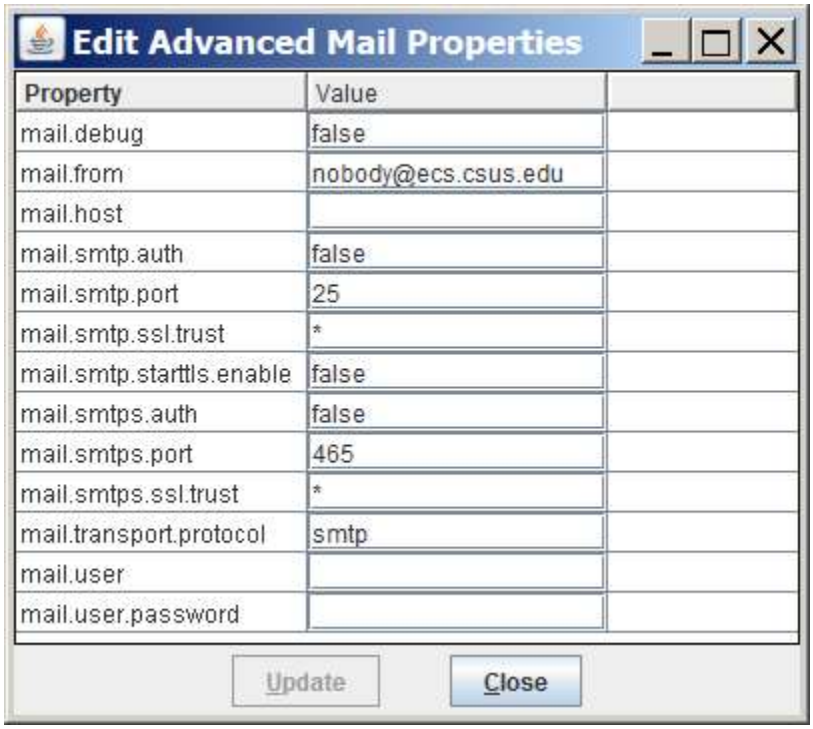
To enable the use of printed balloon notifications for the specified site, check the "**Print Notifications**" box, then enter in the **Print Device** textbox the device identifier of a printer accessible to the PC^2 Scoreboard machine.

Generation of balloon notifications is handled by the PC^2 Scoreboard machine.[30] Once email and/or printing notification is enabled, every "YES" judgment detected by the PC^2 Scoreboard account selected in the "**Balloon Client**" drop-down list will cause an email notification and/or a printed notification to be sent to the configured location.

[30] More specifically, it is handled by a PC^2 Scoreboard machine logged in under the PC^2 account selected as the "Balloon Client" on the Add Notifications Settings screen, as shown above.

6.6.2 Email Server Advanced Settings

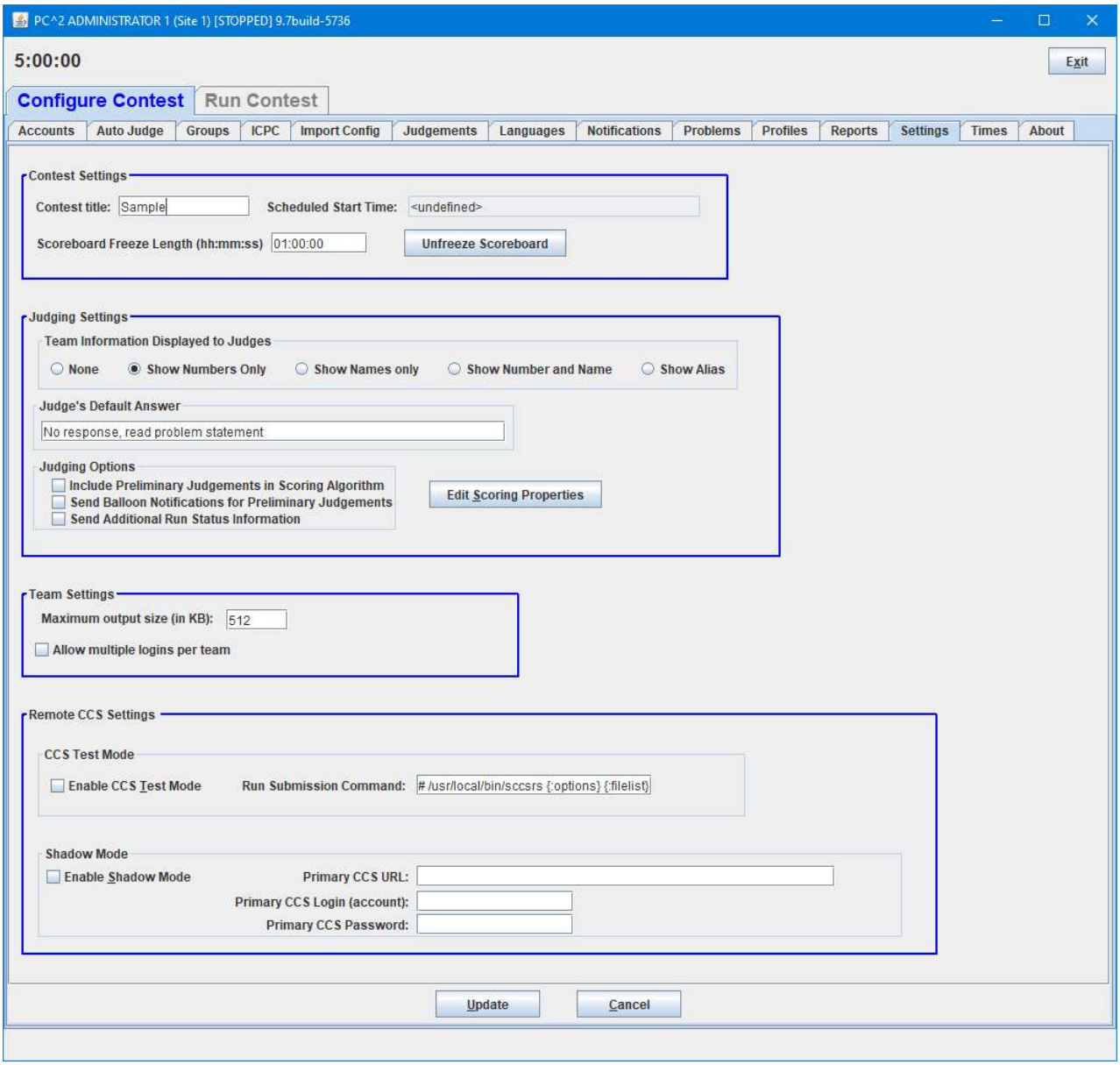
If there are non-standard SMTP or additional SMTP settings required the **EMail Server Advanced Settings** button can be used to specify the settings. This button pops up the following dialog:



The "Value" fields in this dialog are used to configure PC^2 properties related to advanced SMTP operations, including SMTP Authentication. (SMTP Authentication may require some additional external setup to support **smtps** (SSL)). If there are questions about how to use Advanced SMTP properties, including SMTP Authentication, please send email to the PC^2 team (pc2@ecs.csus.edu).

6.7 Options (Settings tab)

The **Settings** tab on the main Administrator **Configure Contest** tab displays a screen similar to the following. The **Settings** tab contains sections which allow selection of various options which can be used manage various aspects of a contest. The meanings of the different configuration fields on the **Settings** tab are given below.



Contest Settings section:

- **Contest Title** – specifies the contest title which is to be displayed on the scoreboard.
- **Scheduled Start Time** – shows the date and time at which the contest will (automatically) start (that is, the date/time at which PC^2 will begin accepting team submissions). The **Edit**

Start Schedule button on the Admin's **Configure Contest > Times** tab is used to enter a scheduled start time. If no scheduled start time has been set (which is the default) then this field displays "**<undefined>**" and the contest will not start until the **Start** or **Start All** button on the **Configure Contest > Times** tab is pushed.

- **Scoreboard Freeze Length** – the point in time (in HH:MM:SS format) *prior to the end of the contest* when the public scoreboard will automatically become "frozen". The public scoreboard remains frozen (meaning any new submissions are shown as "Pending") starting at this point, and remains frozen until the contest *ends* and is "*finalized*". See the chapter on **Scoreboards** for additional information about scoreboard freezing.
- **Unfreeze Scoreboard** – pressing this button will "unfreeze" the public scoreboard; that is, it will allow the public scoreboard to display final contest standings (instead of showing "Pending" submissions) *once the contest is over and finalized*. Note that unfreezing does not take effect until the contest *ends* and has been "*finalized*", regardless of when the button is pressed.

  The normal usage of this button is that the public scoreboard remains frozen (hiding the final results from the public) until the contest ends, at which time the Contest Administrator certifies the final results using the Admin **Run Contest > Finalize** tab and *then* "unfreezes" the public scoreboard using this button. Unfreezing the public scoreboard *cannot be undone*; be *sure* you want to publish the final standings before pressing this button! See the chapter on **Finishing the Contest** for further information about "Finalizing".

Judge Settings section:

- **Team Information Displayed to Judges** – specifies whether or not to reveal the identity of Teams to the (human) Judges while a run is being judged. For example, assume that team 5 has a display name of "CSUS Hornets" and an alias[31] of "Team Orange". Then choosing the various display options will have the following effect on the information shown to the judges:
  - None – show "***" for team name
  - Show Numbers only – show "team#" for team name; for example, "team5"
  - Show Names only – show display name only; for example "CSUS Hornets"
  - Show Number and Name – for example "5 CSUS Hornets"
  - Show Alias– show an alias for the team name; for example, "Team Orange"

  The purpose of this setting is to allow the Contest Administrator to control the amount of information which judges are allowed to know about the team whose submissions they are judging.
- **Judges' Default Answer** – specifies the clarification answer sent to teams if a judge selects the **Default Answer** button while answering a clarification.

[31] To load/specify team aliases see the section **Loading Account Data**

- **Judging Options** – allows the Contest Administrator to control how PC^2 manages certain scoring information; for example, whether to include "Preliminary (Computer) Judgements" in scoring, whether to send notifications for Preliminary Judgements, etc.
- **Edit Scoring Properties** – this button pops up a dialog which is used to specify the scoring point penalties. See the section **Configuring Scoring Properties** in the chapter on the **PC^2 Scoreboard** for additional details.

Team Settings section:

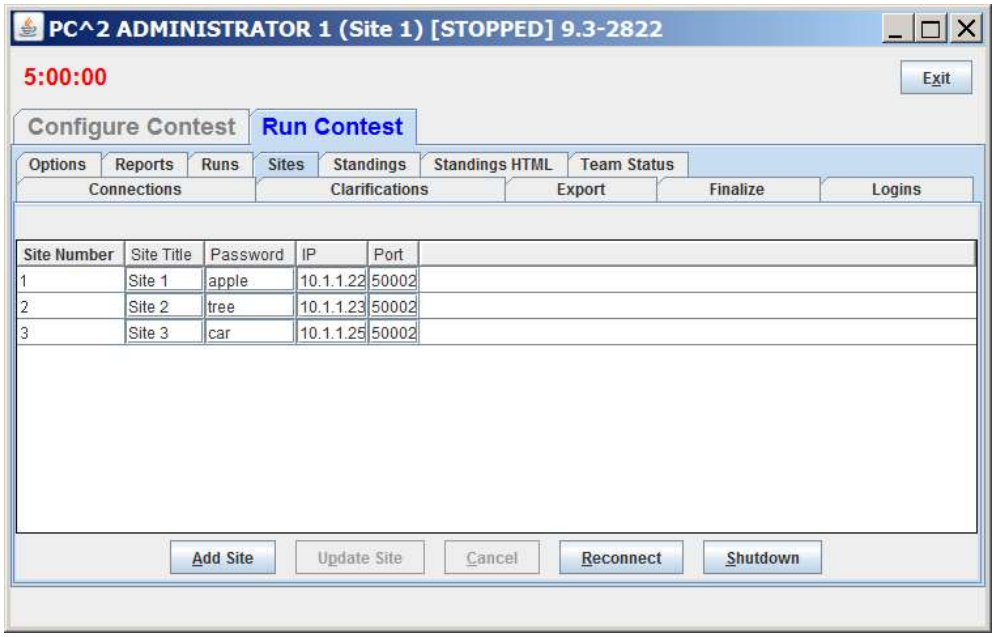
- **Maximum Output Size** - Specifies the maximum amount of output, in Kbytes, which a team program is allowed produce to *stdout* or *stderr*. Any output beyond this amount is discarded by the system and a message is added to the end of the output. The default value is 512 K (1/2 MB).
- **Allow multiple logins per team** – by default, a second login to any account will automatically disconnect the previous login session. If the Contest Administrator checks this box, PC^2 will allow multiple simultaneous logins to *team* accounts (but not to other types of accounts).

Remote CCS Settings section:

This section is used to manage special operations supported by PC^2. See the Appendix on **Shadow Mode** for details.

6.8 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^2 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. Note that the **Sites** tab is on the **Run Contest** tab of the Administrator main screen, not the **Configure Contest** tab.



7 Configuring the Contest via Configuration Files

As an alternative to configuring a contest using the interactive steps outlined in the previous chapter, PC^2 also supports configuring a contest by loading a set of *configuration files*. The ability to load contest configuration files is based on the structure defined by the *CLICS Contest Data Package (CDP)* specification.[32]

The CLICS CDP specification defines a hierarchical directory (folder) structure that includes a sub-directory named "**config**" which holds contest configuration information. The **config** folder contains three files containing configuration information: **contest.yaml**, describing general contest information; **problemset.yaml**, containing general information about the contest problem set; and **system.yaml**, describing contest environment information such as the programming languages supported in the contest. Each of these files contains configuration information written in YAML (Yet Another Markup Language).[33] The details of the YAML configuration elements for each of the three configuration files can be found on the PC^2 Wiki at https://pc2.ecs.csus.edu/wiki/Configuring_A_Contest.[34]

The **config** folder also contains one sub-folder for each contest problem defined in the **problemset.yaml** file. Each sub-folder holds configuration information for one specific contest problem, in a format defined by the CLICS *Problem Format* specification.[35]

Configuration files can be loaded into PC^2 in two different ways: by specifying "**`--load`**" on the server startup command line, or by using interactive controls on the **Import Config** tab of the Admin **Configure Contest** screen. These are described in the following sections (there are slight differences in the results of each method).

7.1 Loading Configuration Files via the PC^2 Server

When loading configuration files using the "**`--load`**" option on the server startup command line, the "**`--load`**" must be followed by a full path ending in either a file name or a directory (folder) name. If a path to a directory is specified, PC^2 expects that directory to be either the root of a CDP or else the **config** folder beneath a CDP root. In either case it reads the contest configuration (including **contest.yaml**, **problemset.yaml**, **system.yaml**, and each of the Problem Format sub-folders) from the specified CDP **config** folder.

If the argument following "**`--load`**" is not a directory, PC^2 expects it to be the full name (including the path) to a file named "**contest.yaml**", and expects that **contest.yaml** file to be located in the **config** file of a CDP. In this case it reads the contest configuration out of the specified **config** folder as above.

---

[32] The CLICS CDP specification can be found at https://clics.ecs.baylor.edu/index.php/CDP, and is also discussed on the PC^2 Wiki at https://pc2.ecs.csus.edu/wiki/Contest_Data_Package.

[33] Or, "YAML Ain't Markup Language", depending on your source and timeframe…

[34] Additional information on the YAML configuration file format can also be found at https://clics.ecs.baylor.edu/index.php/Contest_Control_System#Appendix:_File_formats.

[35] See https://clics.ecs.baylor.edu/index.php/Problem_format for details on the Problem Format structure.

Loading a contest configuration using **`--load`** is a one-time operation. That is, if a server is restarted with a **`--load`** option after a configuration has already been loaded it will display an warning message like "Warning: contest configuration already exists; ignoring --load option" and will ignore the attempt to re-load a configuration (this is done to avoid overwriting a configuration once it has been loaded, because the configuration might have been changed by the Admin after it was loaded). To avoid this (that is, to load a changed configuration), see the following section.

7.2 Loading Configuration Files via the PC² Admin

Configuration files can be loaded interactively (as opposed to using "**`--load`**" on Server startup) by pressing the **Import contest.yaml** button on the **Import Config** tab of the Admin **Configure Contest** screen. Pressing **Import contest.yaml** displays a "file navigation" dialog allowing the user to select a **contest.yaml** file. The selected file is assumed to reside the **config** folder beneath a CDP root; PC² then loads the configuration as described above for the server "**`--load`**" option.

One difference in loading a contest configuration via the **Import contest.yaml** button (as opposed to using "**`--load`**" on Server startup) is that the newly-loaded configuration will be *merged* with the existing contest configuration. That is, if there are conflicts between the existing configuration and the configuration found in the CDP selected via the **Import contest.yaml** button, the system will display the differences and ask the user to confirm which configuration values to use.

7.3 Additional Configuration File Capabilities

With a few exceptions, PC² supports all of the YAML contest configuration items defined by the **contest.yaml**, **system.yaml**, and **problemset.yaml** subsections of the *Input Files* section specified at https://clics.ecs.baylor.edu/index.php/Contest_Control_System#Input_files. (The exceptions include that YAML import specifications for Default Clarification and Clarification Categories will be ignored in the current system (we're working on it…)).[36]

PC² also supports a variety of additional YAML configuration capabilities beyond those required by the CLICS specification (the CLICS specification lists minimum requirements but does not prohibit compatible extensions). The additional supported configuration items are listed on the PC² Wiki at https://pc2.ecs.csus.edu/wiki/CCS_Enhancements.

Further, PC² supports the ability to import a list of passwords which can then be used by the internal account generation process of the system; this is done via the **Import Passwords** button on the **Import Config** tab. The format of the password file is described at http://pc2.ecs.csus.edu/wiki/Passwords.txt. Clicking **Import Passwords** prompts for the name of a password file, then reads the file and assigns the specified passwords to team accounts in the listed order (the first password is assigned to Team1; the second to Team2; etc.).

[36] The PC2Wiki page at https://pc2.ecs.csus.edu/wiki/CCS_Enhancements describes the currently-supported YAML configuration values in more detail.

Imported passwords must appear one per line in the file. If there are more passwords in the file than the number of team accounts, new accounts are automatically created and assigned the additional passwords. Only TEAM passwords can be assigned this way; passwords for Judge and other account types must be set using the interactive methods described earlier. Also, loading a password file only assigns passwords to team accounts at the current site; in a multi-site contest a separate **Import Passwords** operation must be done at each site.

The settings described in the Defining Judging Type section can be defined in the **problem.yaml** file for each problem. Examples and more information about the settings can be found at the wiki article https://pc2.ecs.csus.edu/wiki/CCS_Enhancements#Judging_Types

The current contest configuration can be exported (saved) to YAML files using the **Export Contest YAML Files** option on the **Reports** tab of the Admin **Configure Contest** screen. Note that it is the *current* contest configuration which gets written to YAML files when **Export Contest YAML Files** is invoked; any changes that have been made since loading a YAML configuration will be reflected in the output YAML report (file).

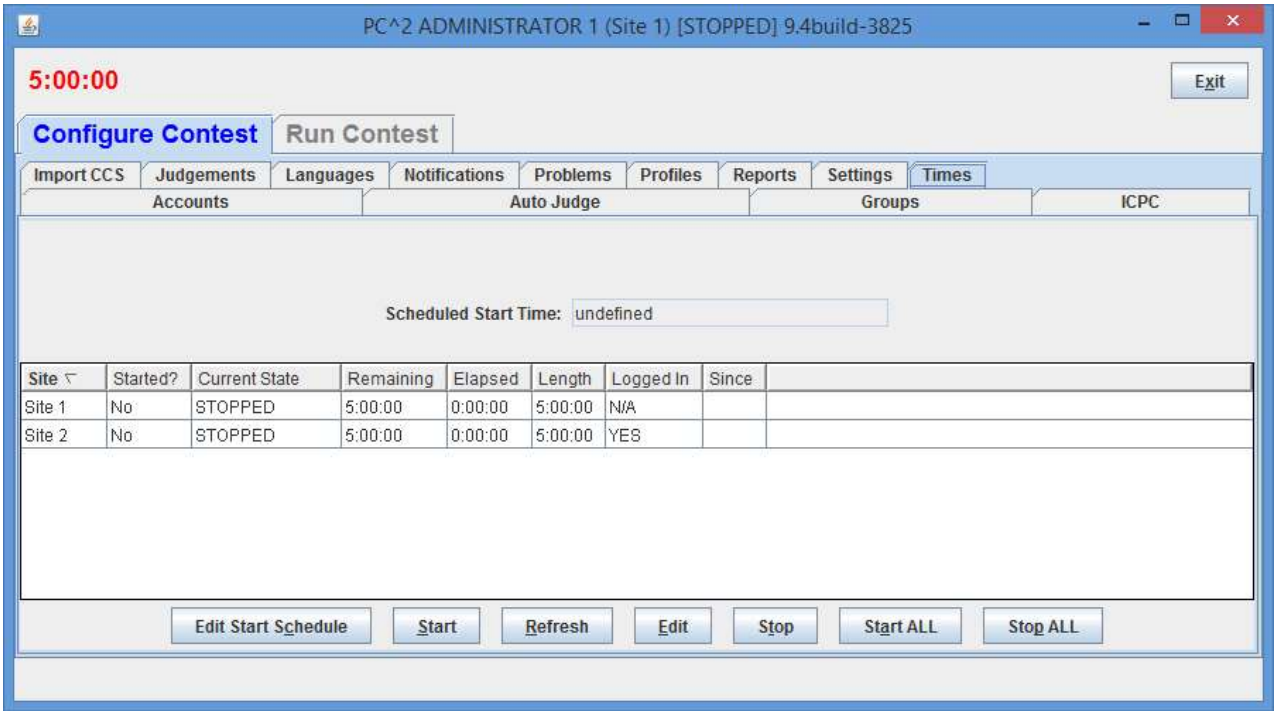
8 Starting the Contest

8.1 Clock Control

Once the contest has been has been fully configured in PC^2, 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 – although it WILL allow teams to log in). As of Version 9.4, PC^2 supports two different methods of starting the contest clock: manual and automatic. (Prior to version 9.4 the only method available for starting the contest clock was manually.)

8.1.1 Starting the Contest Manually

The **Times** tab on the **Configure Contest** tab on the Administrator screen is used to control the contest clock display and to start and stop the contest clock manually. Clicking the **Times** tab produces a screen similar to the one shown below:



The **Start** button is used to tell PC^2 to start the contest clock for a site manually (i.e., now). Selecting a site (by clicking on a row in the table) and then pressing **Start** starts the contest clock running *for that site* and allows teams *at that site* to submit runs. Pressing the **Start ALL** button starts the contest clock at all sites.[37] (See the following section on Multi-Site Clock Control for further information about the **Start ALL** button.)

[37] If the contest clock for some site has already been started when **Start ALL** is pressed, this command causes no change at that site but causes no harm otherwise – it starts the clock at all the other sites.

The amount of time remaining in the contest at the current site is displayed in the top left part of the window just below the window title; the example above shows 5 hours left in the contest. The remaining time automatically starts counting down as soon as the **Start** (or **Start ALL**) button is pressed. It continues to automatically update (count down) as long as the contest clock is running.[38]

The remaining and elapsed time in the grid will not update automatically; to update those times use the **Refresh** button

It is important to note that, from the point of view of PC^2, the contest does not start until the **Start** button is pressed (or the contest clock starts automatically; see below). For this reason **it is important that the Contest Administrator remember to press the Start button at the actual time the contest starts** (or to schedule an automatic start, as described below). 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^2 **Start** 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^2. 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.

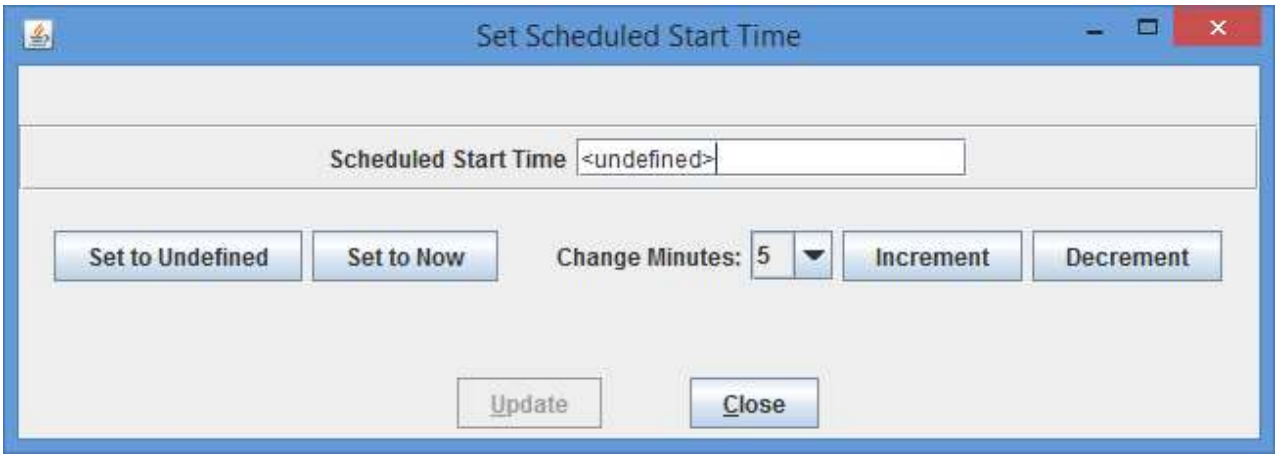
The **Stop** button is used to tell PC^2 to stop the contest clock *for a selected site*. The **Stop** button can be used to insert a pause in a contest (for example, to allow a break for lunch). During the time the contest is stopped, the contest clock at the site does not count down, and teams are prohibited from submitting runs. Also, when the contest is stopped the contest clock displays in RED (as seen above). When the **Start** (or **Start ALL**) button is pushed again, the contest clock 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^2 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.)

8.1.2 Starting the Contest Automatically

As of Version 9.4, the PC^2 contest clock can be scheduled to start automatically at any arbitrary future time. Setting a Scheduled Start Time is done by pressing the **Edit Start Schedule** button on the **Times** screen (above), which produces a dialog similar to the following:

[38] The remaining time is also displayed on the Team and Judge client screens, and counts down automatically; however, it is displayed only to a resolution of one minute on those screens. When the Team or Judge window is minimized the remaining time countdown is displayed in the window icon.



Pressing **Set to Undefined** clears any currently-scheduled automatic start time. Pressing **Set to Now** puts the current time into the **Scheduled Start Time** text box (shown as containing "<undefined>" in the above display). Note however that a Scheduled Start Time must be *in the future*, so merely pressing **Set to Now** then pressing **Update** will generate an error message and request a valid (future) time.

The Scheduled Start Time displayed in the text box can be changed by selecting a value in the **Change Minutes:** dropdown list and then pressing **Increment** (to move the time farther into the future) or **Decrement** (to move the time back – but again, the dialog will not accept a time that is not at least one minute in the future).

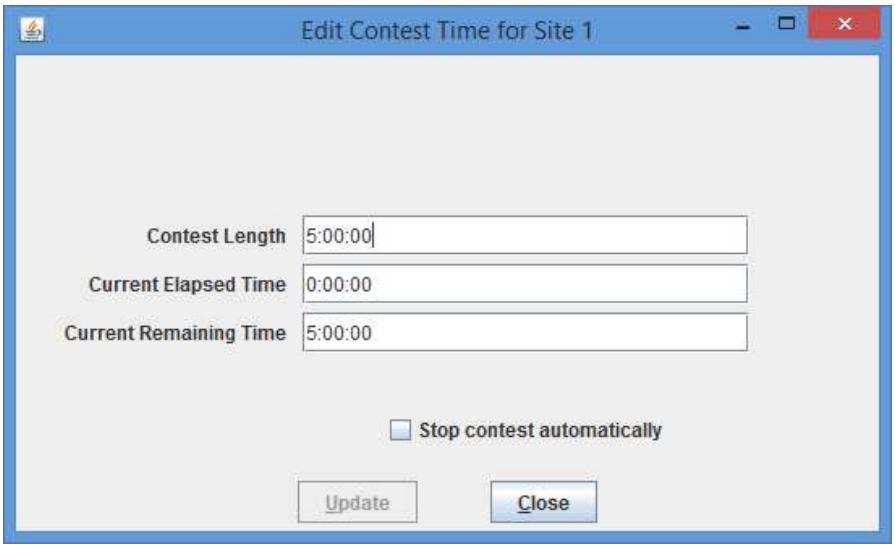
Once a desired valid (future) Scheduled Start Time has been selected, pressing **Update** will cause PC^2 to set up a task to automatically start the contest at the specified time; it will not be necessary to push the **Start/Start ALL** button to start the contest (although doing so will work; doing so will override any scheduled future start time and start the contest when the **Start/Start ALL** button is pushed).

When a Scheduled Start Time has been set it will be displayed in the **Times** screen shown previously, and the contest will automatically start (that is, the clock will start counting down and teams will be allowed to submit runs) when the specified time has arrived. Note that a Scheduled Start Time is interpreted to be *contest wide* and an automatic start starts the clock at *all sites* in a multi-site contest; there is no facility supporting "automatic start" for different sites at different times (manual starting must be used to do this).

Note: automatic contest starting is also supported in PC^2 Version 9.4 and above via the "**/starttime**" web service. See the Appendix on Web Services for further details.

8.2 Contest Length

The **Edit** button on the **Times** tab displays the **Edit Contest Time** dialog (shown below) which allows the Administrator to change the contest length, elapsed time, or remaining time.



The time values in the **Edit 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 already stopped when the dialog is activated, those times remain accurate indefinitely.

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.

The "**Stop contest automatically**" checkbox will automatically stop the contest clock when the remaining time reaches 0:00. Unlike the other contest time settings which are a local site setting, this setting applies to every connected site. If the checkbox is **not** checked then teams can still submit runs. Any runs that are submitted after the end of contest (where the elapsed time is greater than contest length) will be marked as DEL (deleted) and those runs will not be shown on the scoreboard or event feeds. If the contest clock is running after the end of the contest the contest time is preceded by a +. For example if the contest is running for 25 seconds after the end of the contest the contest time would display as +0:00:25.

8.3 Multi-Site Clock Control

As described above, the contest clock in PC^2 operates on a per-site basis. That is, each site in a multi-site contest has its own contest clock, and PC^2 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).

Normally, the **Start All** and **Stop All** buttons should be used to start/stop the contest clock at the same moment in time at all sites. The ability to start each site separately is a convenience mechanism intended to support situations where one site is "ready to go" and the Contest Director does not wish to delay that site while waiting for other sites to become ready. Having separate

contest clocks for each site allows each site to operate quasi-independently while insuring that contest results are calculated correctly (meaning that teams get credit for submitting runs based on the contest time *at their site*, and the PC^2 Scoreboard takes differences in site clocks into account when computing standings). However, operating the clocks separately at each site has its risks, as described below.

Each PC^2 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** 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**" button on the **Times** tab. This button performs the same function as **Start** 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).[39]

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** button is the preferred (safest) way to coordinate the start of a contest. Likewise, if all sites are tightly coordinated the **Stop All** 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

[39] If a new site server is started after the **Start All Sites** button is pressed, it will be necessary to use an Admin client to start the contest time for that site. Alternatively, you could use **Start All Sites** to start all sites including the new site.

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).

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^2 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^2 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^2 at that site to have been submitted "1 minute into the contest".

Caveat Administrator:

It is *critical* that the PC^2 "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 contest automatically** checkbox is a contest wide setting. If a site is connected when the checkbox is checked then that site will stop the contest clock whether the site is connected or not, each countdown to stop the contest clock starts a local Thread.

8.4 Practice Sessions: Resetting A Contest

NOTE: with PC^2 Version 9.3 and above, much of what is described in this section can be done much more cleanly using *Profiles*; see the section on **Contest Profiles** for additional information.

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^2 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^2 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^2 "clarification system" during the practice contest. A practice session also has the advantage of giving the Judges practice with how PC^2 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^2 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(s) 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, judgments, 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. In addition, it is necessary to switch from the practice problem set to the real contest problem set.

There is a relatively simple way to accomplish a switch between a practice and a real contest while preserving the necessary information. First, create two directories named (for example) **`practice`** and **`real`**. Change to the **`practice`** directory, start a PC^2 Server in that directory, and then start an Admin and configure a contest including the team accounts, languages, etc., but *omitting the problem definitions*. Next, shut down the server then (recursively) copy the **`practice`** directory contents to the **`real`** directory. This creates two directories with identical contest configurations, including accounts, languages, etc., but with no contest problems.

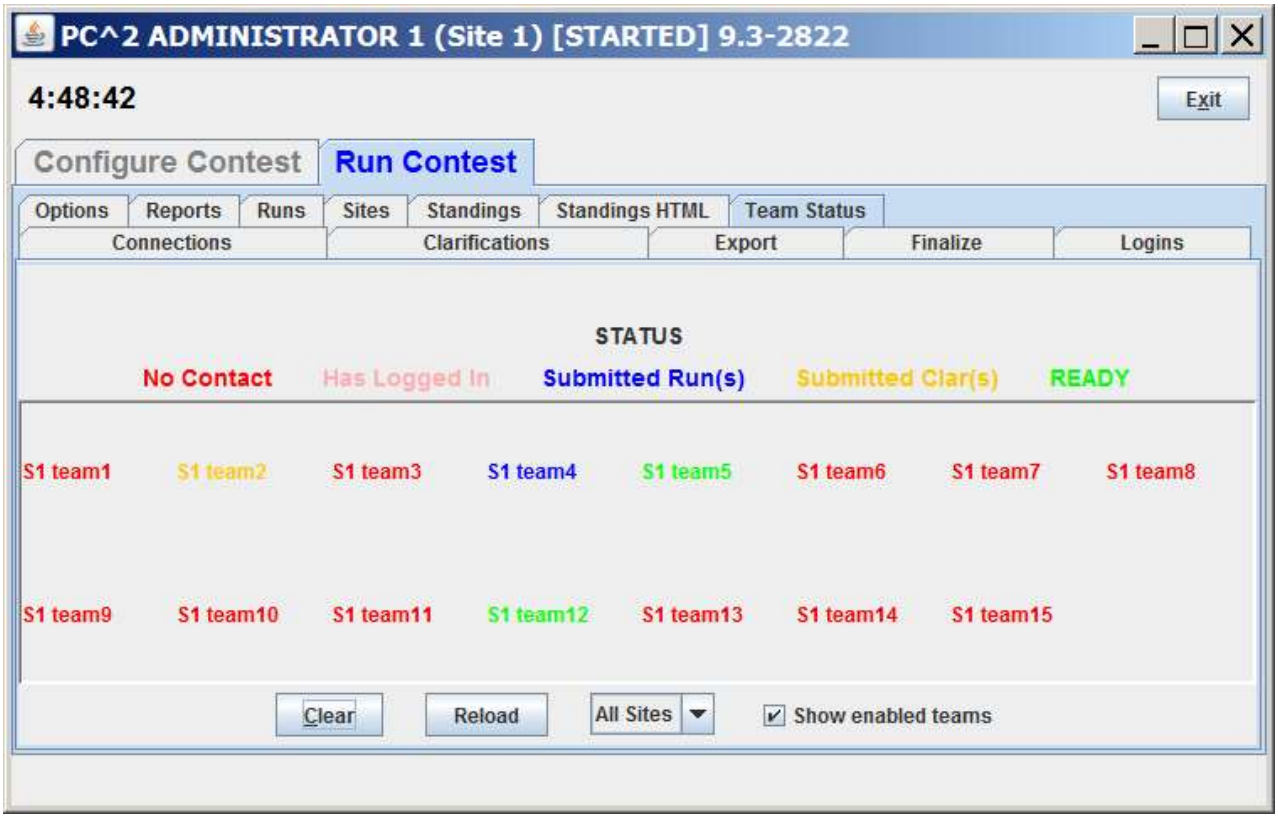
Next, restart a server in the **`real`** directory and then run an Admin and add the real contest problems to that configuration. Finally, shut down the server, change to the **`practice`** directory, and restart a server and an Admin in that directory and use the Admin to add the practice contest problem definitions to the **`practice`** contest configuration. This way, the real and practice contests both have the same accounts, languages, etc., but have different problem sets. To switch to the real contest, shut down the practice contest server, switch to the **`real`** directory, and start a new server in that directory.

9 Monitoring Contest Status

9.1 Team Startup Status

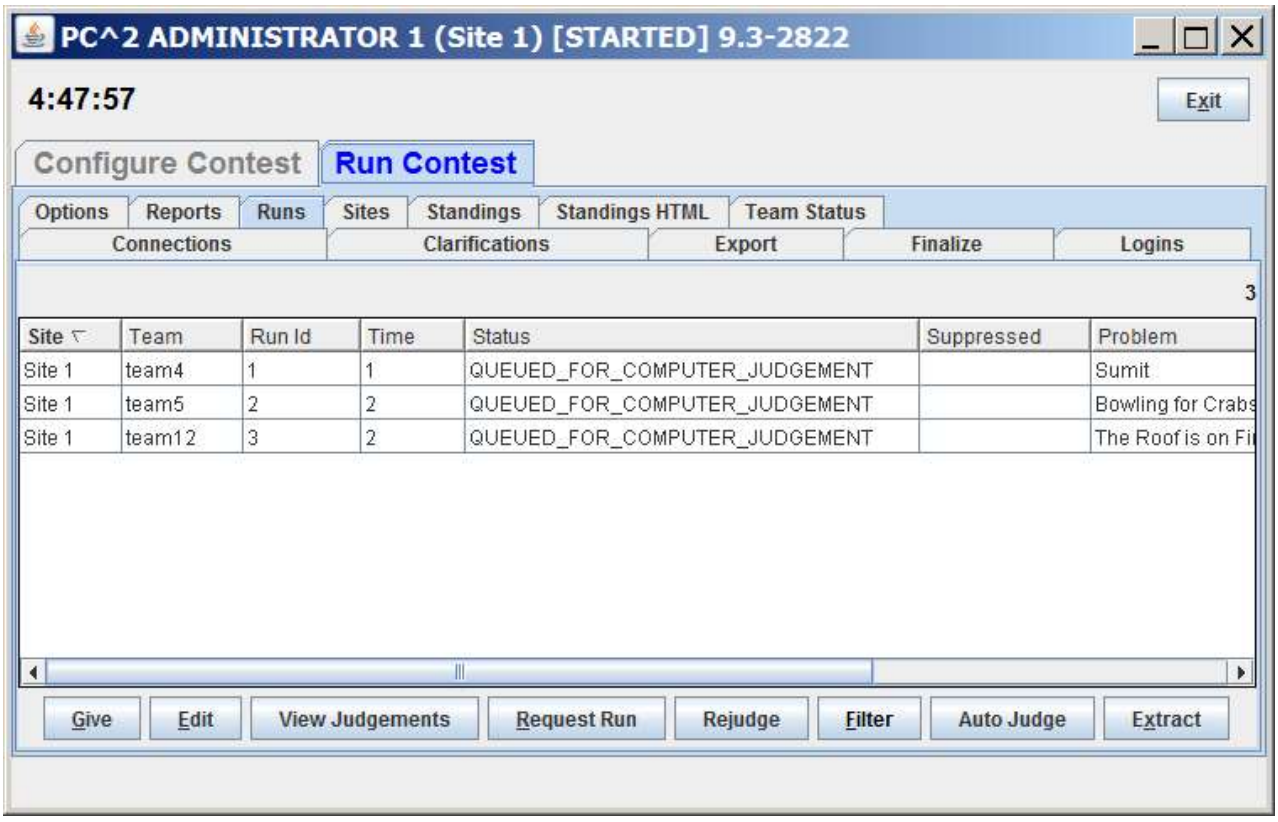
The **Team Status** tab on the main Administrator screen Run Contest tab is used to track the status of Teams once a contest has been started. This is particularly useful during a "practice contest" held just prior to a real contest; it allows the Contest Administrator to verify that all teams have been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

A sample **Team Status** screen is shown below. Initially all teams are displayed in RED, indicating that the team has not made any contact with the PC^2 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^2 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.)



9.2 The Runs Display

The **Runs** tab on the Run Contest tab on the 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.

9.3 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.[40]

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:

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

[40] 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^2 is not smart enough to decide whether you *should* change the attributes of a run; it just gives you the *capability* to do so.

change multiple attributes of a run during a single edit (although this would be unusual; normally, editing a run is done for a single 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 "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.

Checking the "Delete Run" 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. A run marked as deleted no longer shows on the appropriate Team's grid. ("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 "Notify Team" 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 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^2 does not keep track of the old run state once the **Update** button has been pushed).

9.3.1 Extracting Runs

The **Extract** button on the Edit Run dialog allows the run being edited to be "extracted" (exported) to a separate directory. When the Extract button is clicked the following will appear:

The extract directory is located under the $PC2HOME directory. If Run 2 for Site 1 was being edited and the Extract button was pushed, the following files would be created in the extract directory:

```
extract/site1run2/pc2.run2.txt
extract/site1run2/Prac.java
```

The `pc2.run2.txt` file contains information about the run including Run #, Site #, Team Id, Problem Name, Language Name and contest elapsed time for example:

```
...
Run   : 2
Site  : 1
Team  : TEAM5 @ site 1
Prob  : Bowling for Crabs
Lang  : Java
Elaps : 2
...
```

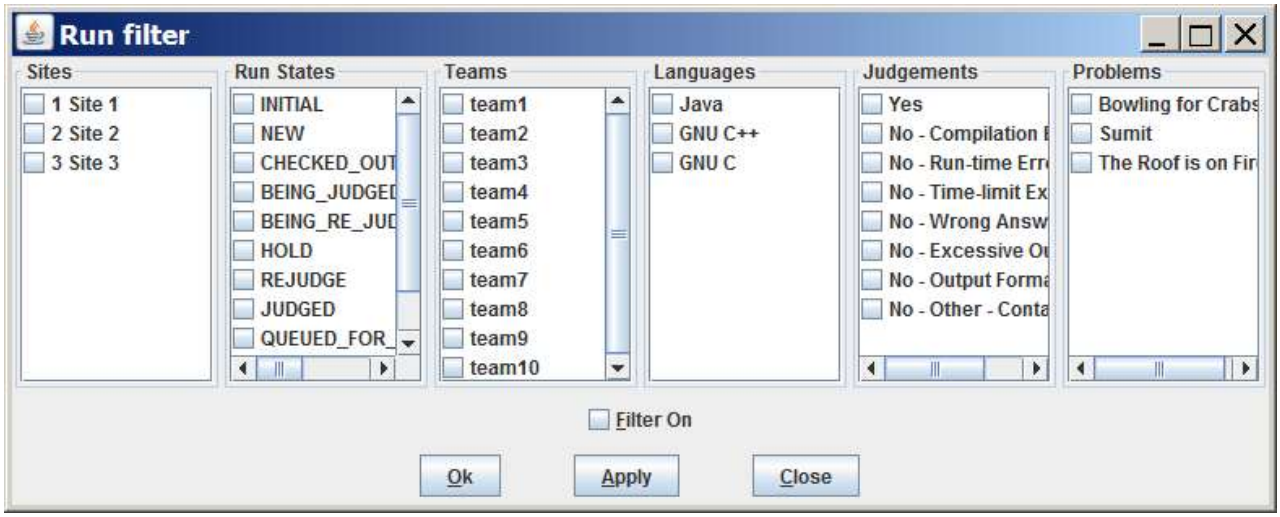
The second file (`Prac.java` in the example above) contains the source code which was submitted for the run which has been extracted.

The current version of PC^2 has no way to extract multiple runs with a single action; to extract several runs it is (unfortunately) necessary to Edit each run and extract it individually. (Yes, fixing this is on our list of "to-do's").

9.4 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.[41] Pushing the **Filter** button on the **Runs** display screen will produce the following dialog, which is used to set the filtering criteria:



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 "Site 1" for the problem named "Sumit", you would select "Sumit" in the Problems column, "team1" in the Teams column, "1 Site 1" 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.

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.

The "Ok " field is used to specify the command line which is used to compile source code and produce an "executable program file" in the language.

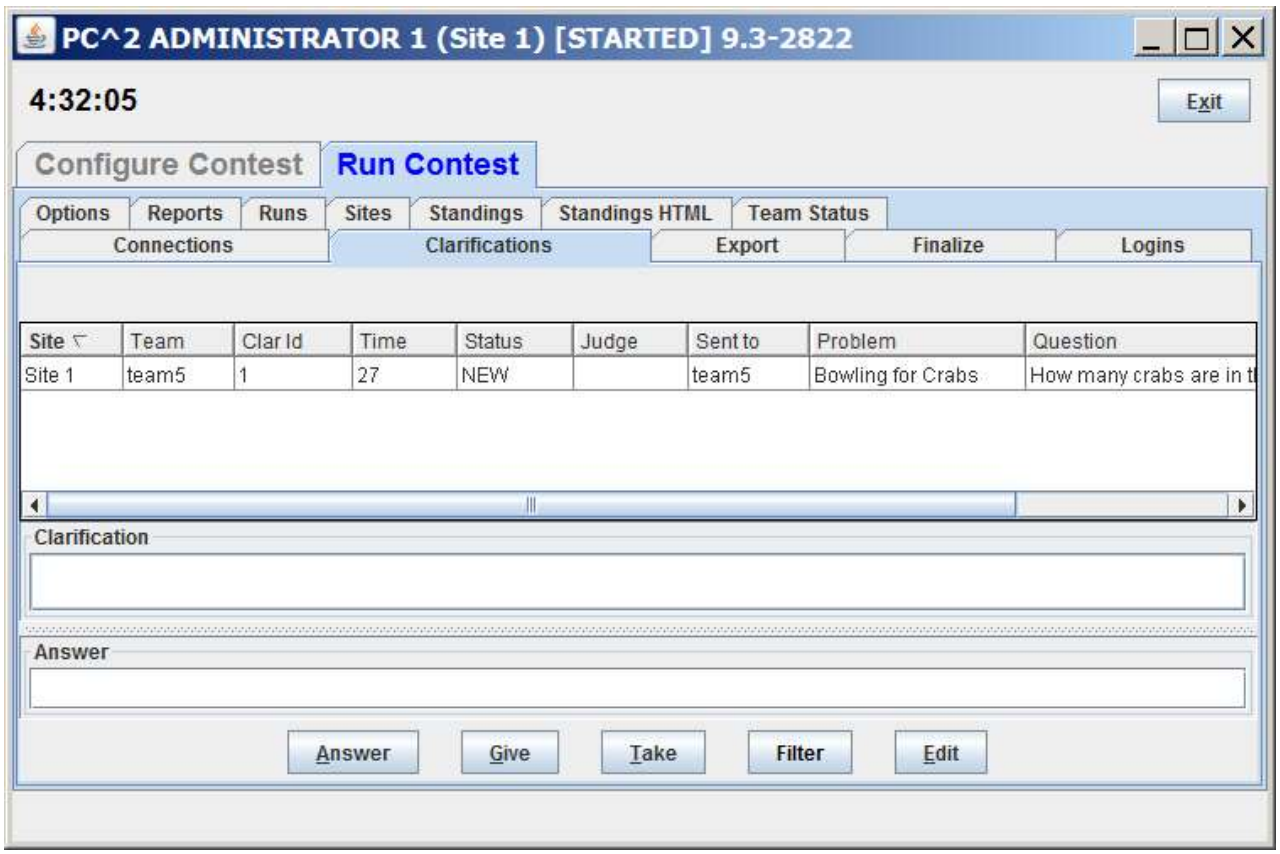
The **Ok** button will save the filter settings and if the filter is ON will update the Runs list with the filter settings.

[41] The Judge client provides a similar filtering operation; the discussion here on filters applies equally to the Judge.

The **Apply** button will immediately reload the Runs list with the filter settings (if Filter On is checked). This can be used to quickly find runs based on the filter instead of having to use 3 steps to see the filtered results. Three steps would be: 1) (Edit) Filter, 2) change filter 3) **Ok** button

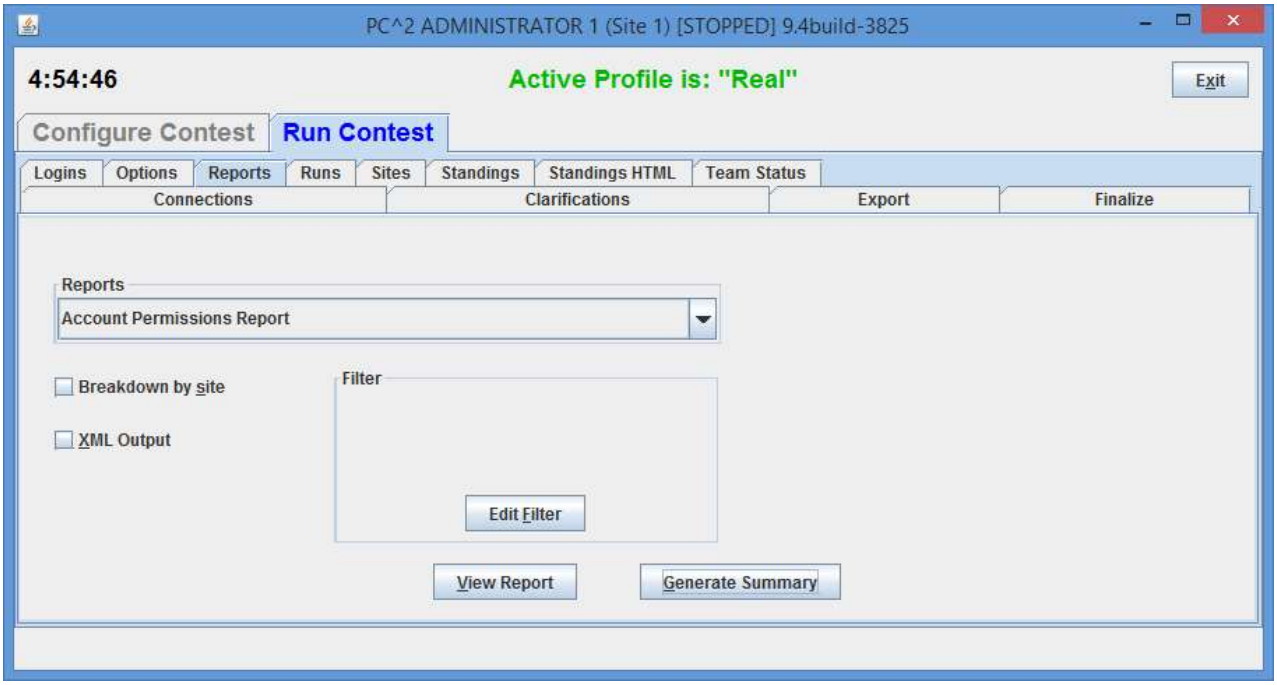
9.5 Clarifications

The **Clarifications** tab on the main Administrator screen **Run Contest** tab, shown below, 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** functions work like the **Give** and **Take** on the Runs pane.



9.6 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. Note that the **Reports** tab appears on both the **Configure Contest** and **Run Contest** Administrator tab. The **Reports** screen looks like the following:



The "**Reports**" 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 Report**" 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)

Pressing "**Generate Summary**" will generate all the following reports: Balloons Summary, Clarifications, Contest Analysis, Contest Settings, Contest XML, Evaluations, Fastest Solutions Per Problem, Fastest Solutions Summary, Groups, ICPCTools Event Feed, JSON Standings, Languages, Problems, Runs, Runs (Version 8 content and format), Runs grouped by team, Scoreboard, Solutions By Problem, Standings Web Pages, Standings XML, Submission, and Submissions by Language. When all reports are written a dialog will appear that shows the directory where the reports were saved. Reports will be saved in the `profiles\P<profileid>\reports` directory.

9.6.1 Automatic Generation of Reports at End of Contest

When a server is shut down and there are less than 30 minutes remaining in the contest, the following reports will be automatically generated in the reports directory.

Report	Description
Account Permissions Report	For each client list their Permissions/Abilities
Accounts	List summary of accounts per site, and individual accounts sites, logins, passwords.
All Reports	List contents of all reports
Balloons Delivery	List of all balloon deliveries by team, by problem and time of delivery
Balloons Summary	List summary of which teams should have which color balloons.
Clarifications	List all clarifications
Client Settings	Various settings like Notification settings
Contest	Contest settings in XML (work in progress)
Contest Analysis	Summary of submissions, unjudged runs, various checks on runs.
Evaluations	One line per judgment output
Extract Replay Runs	Files extracted used with Replay feature
Fastest Solved by Problem	List all run solving problems by fastest, by problem, and fastest solution showing rank, elapsed, team name
Groups	List of Groups (Regions)
Internal Dump	An internal dump of a bunch of config settings
Judgement Notifications	List (balloon) notifications grouped by problem.
Judgements	List of judgments
Languages	List Languages
Logins	List who is logged in
Notification Settings	List of Notification Settings
Problems	List problems
Run 5 field	List of runs: run #, team #, problem letter, elapsed time, judgment
Run Notifications Sent	
Runs	List of runs, with run#, run state, team #, team name, whether judgment sent to team and details on each judgment

Report	Description
Runs (Version 8 content and format[42])	List of runs with detail (see below)
Runs grouped by team	List of runs, grouped by team, then by problem, helpful in calculating scoring.
Solutions By Problem	For each problem show number of run with No, Yes, and percentage correct
Standings XML	Standings in XML format
Submissions by Language	A summary of how many teams used which languages.

9.7 Event Feed

PC^2 complies with the CLICS Contest Control System specification (found at https://clics.ecs.baylor.edu/index.php/Contest_Control_System). Among other things this means that it generates an *event feed* providing external tools with real-time notification of events which occur in the contest.

Note in particular that this means that PC^2 is compatible with the **ICPC Tools** suite of contest utilities. The PC^2 Event Feed can be used for example to drive the ICPC Tools *Resolver* to produce dynamic displays revealing the final results of the contest (at an Awards Ceremony, for example). It can also be used to drive the ICPC Tools *Balloon Utility*, an interactive tools for managed balloons during a contest, as well as a variety of other contest utilities available in the ICPC Tools.

For details on the PC^2 Event Feed mechanism and how to use it, see the Appendices. For information on the ICPC Tools suite of contest utilities, visit https://icpc.baylor.edu/icpctools/.

9.8 Web Services

PC^2 contains an embedded web server which provides a variety of REST-based web services. Currently-implemented services include resources for obtaining the current list of contest teams, problems, languages, and the CLICS 2016 JSON Scoreboard. Also included are REST services for managing automated (scheduled) contest start time. For details on the PC^2 REST Services and how to access them, see the Appendices.

[42] Example single line of output: run 59|site 1|proxy |team 11|team11:Crimson Pride (WSU)|prob D - Obstacle Course-6938491730217106684:D - Obstacle Course|lang GNU C++-1782097724418977383:GNU C++|tocj |os Linux|sel false|tocj false|jc true|182|rid Run-2641274401129060175|mmfr true|del? false|jt 0|jby judge5|jci Yes|

10 The PC² Scoreboard

10.1 Overview

PC² 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[43] (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 into the ICPC Contest Management System (CMS)).

Some contests are run by displaying the actual, current contest standings on a scoreboard throughout the duration of the contest. However, many contests are run using the concept of a "scoreboard freeze" – that is, a period of time near the end of the contest when the judge's responses to team submissions (and the effects of those responses on the contest standings) are hidden, to be revealed at a post-contest ceremony. During such a "scoreboard freeze period" any submissions received from teams are displayed on the scoreboard as "Pending", and the results of those submissions (and their effects on the contest standings) remain hidden.

The PC² Scoreboard module supports generation of two different sets of HTML files: one set which respects any configured scoreboard freeze (referred to as the *public* or *freezable* scoreboard), and a second set which shows the current actual contest standings, *ignoring any "freeze period"*. In a contest utilizing a scoreboard freeze, the first (freezable) set of HTML files is appropriate for "public viewing", while the second set is intended for "private" viewing only by authorized contest personnel.

The public and private scoreboard HTML files are automatically written into two separate (configurable) folders; see the section below on **Configuring Scoring Properties** for details. Managing the settings for the time in the contest at which the public scoreboard becomes "frozen", and for when it subsequently becomes "unfrozen" so that the final standings are publicly viewable, is controlled by settings on the Admin **Configure Contest > Settings** tab (see the section on **Options** in the chapter on **Interactive Contest Configuration** for further details).

10.2 Scoring Algorithm

The algorithm used in PC² to compute Rank and "Score" (Penalty Points) is the one used in the 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

[43] While PC² contains its own internal "Balloon notification" capabilities, since Version 9.5 it is compatible with the ICPC Tools "Balloon Utility" (see https://tools.icpc.global/). Because the ICPC Tools Balloon Utility is considerably more robust and flexible, we recommend using that tool to support management of balloons.

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:

- Some number of penalty points 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).
- Some 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).[44]

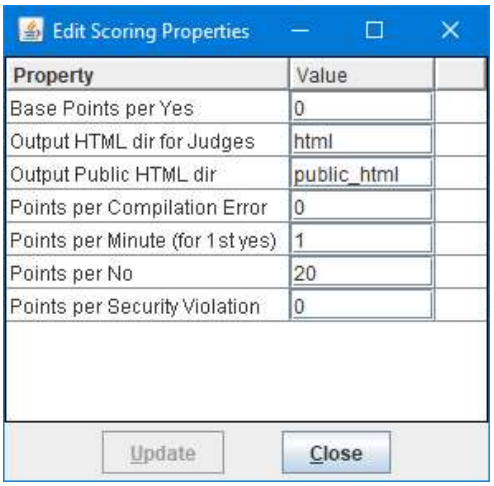
3) If two or more teams have the same number of solved problems and exactly the same number of Penalty Points, ties are broken in favor of the team with the earliest time of the last correct submission (that being the time when the team "finished" the contest).

4) If two or more teams remain tied for a given rank after all the above criteria have been applied, the teams are listed in alphabetical order within their rank by the team's "display name".

Examples of the PC^2 scoring algorithm can be found in the PC^2 Wiki article Scoring Algorithm at http://pc2.ecs.csus.edu/wiki/Scoring_Algorithm.

10.3 Configuring Scoring Properties

Certain properties used in scoring are configurable by the Contest Administrator. To adjust scoring properties, select the **Settings** tab on the Contest Administrator[45] **Configure Contest** tab, then press the **Edit Scoring Properties** button. This will bring up the following dialog:



[44] In the ICPC World Finals the number of penalty points for each incorrect submission prior to solving a problem is always 20 and the number of penalty points per minute until the problem is solved is 1; these are the default values in PC^2 (although PC^2 allows the Contest Administrator to change these values).

[45] Note that while Scoring Properties control the computations performed by the Scoreboard module, for security reasons they are only editable on the PC^2 *Admin*.

The Scoring Properties have the following meanings:

- **Base Points per Yes**: the number of penalty points, in addition to **Points per Minute (for 1st yes)**), which are always assigned to submissions which correctly solve a problem.

- **Output HTML dir for Judges**: the name of the folder where the scoreboard will output HTML files showing the actual current contest standings. This output *ignores any "scoreboard freeze"*; it shows the actual current contest standings. Typically the contents of this folder would be made available only to Judges and other privileged contest personnel.

- **Output Public HTML dir**: the name of the folder where the "public" scoreboard HTML files will be placed. The data written to this folder will respect any "scoreboard freeze" that is configured. That is, it will not show results for any submissions sent in during the "freeze period", but rather will show any such submissions as "Pending". Typically the contents of this folder are made available to the general public as the "Current Standings".

- **Points per Compilation Error**: the number of penalty points assigned to submissions which fail to compile.

- **Points per Minute (for 1st yes)**: the number of points, for each minute elapsed in the contest, assigned to submissions which correctly solve a problem – from the start of the contest until the time of the correct submission. (Note that only the FIRST correct submission is counted; if a team subsequently submits another run for the same problem which correctly solves the problem, no points are assigned based on this category.)

- **Points per No**: the number of penalty points which are assigned for each incorrect submission on a problem prior to the problem being solved (but only if the problem is eventually solved; unsolved problems accrue no penalty points).

- **Points per Security Violation**: penalty points added to a submission which generates a PC^2 Security Violation. Note that Security Violations can only be generated if a PC^2 Security Manager ("sandbox") is installed in the system.

Note that all the above scoring properties can also be set using YAML configuration settings; see the chapter on *Configuring the Contest via Configuration Files* for details.

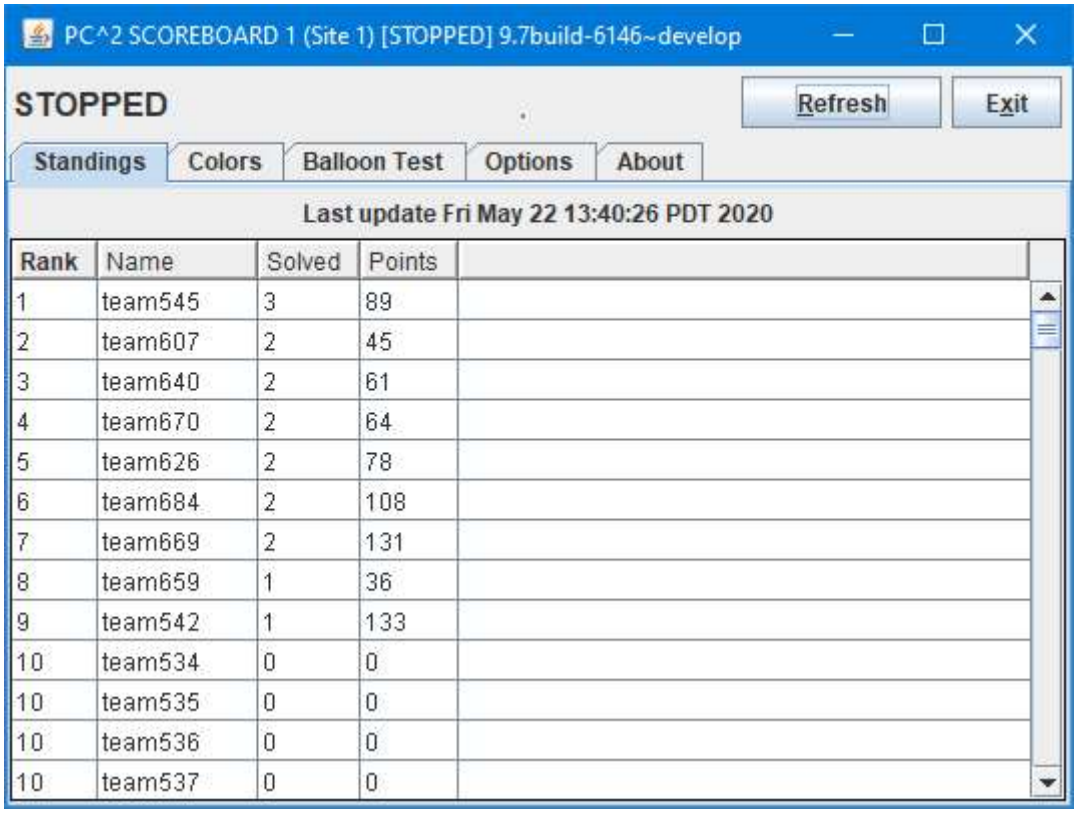
Note also that changing scoring property values only affects that particular scoring criterion; it does not alter the overall determination of ranking criteria. For example, assigning a negative value to one property does not change the fact that teams are ranked first by number of problems solved; a team with a large negative score will still not be ranked higher than a team which has solved more problems.

10.4 Starting the Scoreboard

To start a scoreboard, go to a command prompt in the PC^2 installation directory and type the command "**pc2board**". This will start a PC^2 client expecting a scoreboard login. Once the Client

login window appears, enter the scoreboard account name and password as defined when PC^2 accounts were created.[46]

Logging in to a scoreboard account will bring up a PC^2 Scoreboard display window, similar to the one shown below, indicating that the scoreboard program is running.



Rank	Name	Solved	Points
1	team545	3	89
2	team607	2	45
3	team640	2	61
4	team670	2	64
5	team626	2	78
6	team684	2	108
7	team669	2	131
8	team659	1	36
9	team542	1	133
10	team534	0	0
10	team535	0	0
10	team536	0	0
10	team537	0	0

The scoreboard automatically generates a complete set of public and private HTML files as described above as soon as it is started. Thereafter it generates updated HTML files periodically according to an algorithm described below (see **Scoreboard Updates**). Each time a new set of HTML files is generated the scoreboard display window is updated to show the most recent update time.

Note that *the display generated by the PC^2 Scoreboard module (shown above) always contains the actual current contest standings*. That is, the Scoreboard module display does *NOT* respect any "scoreboard freeze" configured in the contest. Only the HTML files generated in the "public HTML folder" pay attention to freeze period values. For this reason, *the Scoreboard module display should be considered a "private" scoreboard.* (This is another reason for being sure to *change the scoreboard account password.)*

---

[46] Recall that by default, account passwords are the same as the account name. 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.

Under normal circumstances it is only necessary to have a *single* PC^2 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 can generate balloon notifications for all sites. Thus there is rarely a need for running more than one PC^2 scoreboard in a contest, and this is the recommended mode of operation.

10.5 Scoreboard Updates

Once the scoreboard module is running, it waits passively until some contest event occurs which could alter the data it should display. When any such event occurs, the scoreboard obtains an update of the contest state from the server, computes the new standings based on this information, and regenerates the HTML display files. The date and time of the last scoreboard update is shown on the scoreboard module window.

Any of the following events will cause the scoreboard module to generate new HTML files:

- Scoreboard module is started
- Scoreboard module **Refresh** button is clicked.
- Any run is judged
- Any run judgment is changed (via **Edit Run**)
- Contest configuration definitions/settings are changed (accounts, languages, problems, etc.)

10.6 Scoreboard HTML Files

Each HTML file generated by the PC^2 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^2 places the Contest Title as specified by the Contest Administrator (see **Options**, above). Each document **<body>** contains an imbedded **<table>** holding contest status information. Each HTML file is also in an XML formatted document.

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 (see the section on Adding New HTML files for information on how to alter the HTML scoreboard file contents).

The following HTML files are always generated by the scoreboard :

File Name	Table Contents
full.html	Columns showing rank, team display name, number of problems solved, and penalty points, with rows ordered by rank.
fullnums.html	Same as **full.html** except that the Team Display Name is preceded by the Team Number followed by a dash.
sumtime.html	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. The rows in the grid are listed by team number (not rank).
sumatt.html	A table similar to the **sumtime** 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. The rows in the grid are listed by team number (not rank).
summary.html	A table combining the **full** and **sumtime** displays described above – that is, a grid 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.
Index.html	A table showing the same data as **summary** (above), but also including header information showing the Contest Title and a "Scoreboard Message" indicating whether this scoreboard is "Live" (i.e., a "private" scoreboard showing current, actual standings) or instead a "public" scoreboard which will show submissions received during the "scoreboard freeze period" as "Pending". For public scoreboards the "Scoreboard Message" automatically changes from showing "Time until freeze goes into effect" (before the freeze period starts) to "Scoreboard was frozen at <time>" during the freeze period, and to "Final Standings" after the contest has been finalized.

The public and private HTML scoreboard pages are written to *local files* on the machine on which the PC^2 scoreboard is running. One common way to take advantage of the HTML files generated by the scoreboard is to run a separate external process (e.g. a batch script[47]) which repeatedly copies the current *public* HTML files to some external location (web site), 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^2. 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^2 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.

[47] A sample batch script is included with the samples: samps/web/distribute_score .

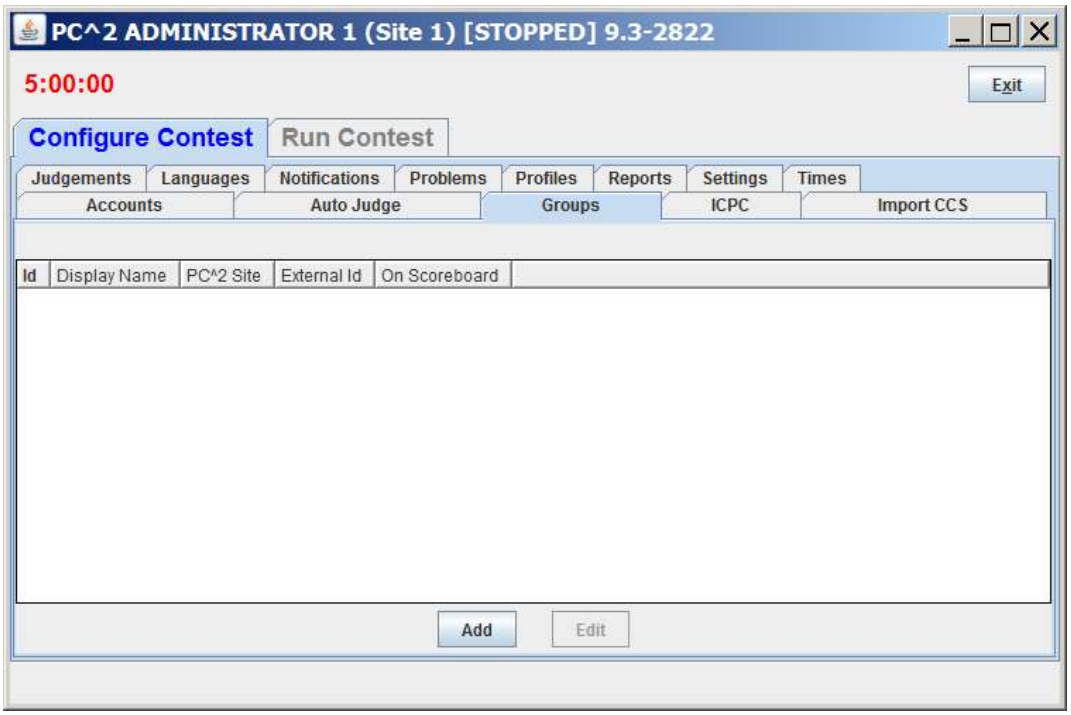
10.7 Scoring Groups

In addition to the above files, the scoreboard can be made to generate separate HTML 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. 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.

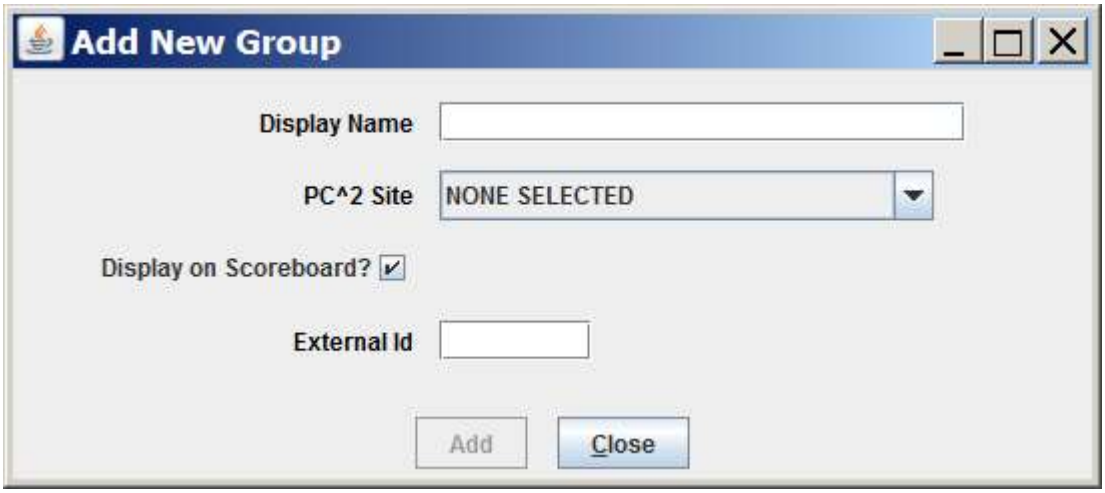
Every PC^2 account has associated with it a "Group" identifying the region or group to which the account belongs. By default accounts do not belong to a group. Changing the group for a particular team account associates that team account with other all teams in the same group.

By default, the scoreboard ignores groups. In order to cause the scoreboard to pay attention to Groups, three steps are required. First, the contest administrator must define the groups to the system. Second, team accounts must be assigned to a group. Third, the contest administrator must provide an XSL file describing how to generate the HTML file representing the group standings (see the section below on **Managing HTML File Generation**).

Groups can be defined either manually or by using the "import" functions on the ICPC tab on the main Contest Administrator screen. To manually define a new group, select the **Groups** tab on the **Configure Contest** tab of the main Administrator screen. This will display a screen similar to the one shown below.



Click the **Add** button to add a new Group. This will display the following **Add New Group** dialog:



In the Display Name field enter a name for the new group (for example, "Upper Division" or "Site 1"). The "**PC^2 Site**" dropdown list and the "External Id" field can be ignored.

If your contest is supported by ICPC Headquarters[48], groups can be set by using the "import" functions on the **ICPC** tab (see the Appendix on **ICPC Import/Export Interfaces**).

Once groups are defined, accounts can be assigned to a group (an account can belong to at most one group). Groups can be set when user accounts are first defined, or by editing accounts later (see the section on User Accounts earlier in this manual).

The third step required to make use of groups is to define the way in which PC^2 should generate HTML output for the groups. This is described in the following section.

10.8 Managing HTML File Generation

PC^2 uses two data sets to create scoreboard output HTML files. The first data set is a pair of eXtensible Markup Language (XML) document strings containing the contest standings – one string for the public (freezable) version, the other for the private (live) version. These strings are automatically generated whenever the standings change, and are cached internally. In addition, the *private* (live) version is written to a file named **results.xml** (overwriting any previous contents) each time the standings change.

The second data set used for scoreboard output file creation is a collection of eXtensible Stylesheet Language (XSL) files describing how the XML standings should be transformed (processed). Whenever the contest standings change, the scoreboard module generates a new pair of standings XML strings, saves the private (live) version to the **results.xml** file, then looks in the *current directory* for a folder named **data** containing a subfolder named **xsl**. If the **data/xsl** folder is

[48] All ICPC Regional Contest Directors have access to the PC^2 ICPC Import Data files describing all the teams registered for their Regional Contest, including their groups ("sites").

present, the scoreboard looks there for a set of XSL files (filenames ending in **.xsl**) to control processing.

If no **data/xsl** folder is found in the current directory, the scoreboard instead looks for **data/xsl** under **$PC2HOME**. The standard PC^2 distribution includes a collection of **.xsl** files – one for each of the standard HTML output files listed above – in the **$PC2HOME/data/xsl** folder.

The scoreboard reads each **.xsl** file found in the **data/xsl** directory and uses the XSL transformations specified in that file to create a new **.html** file in the appropriate (public or private) **html** directory.[49] For example, the presence of a file named **full.xsl** will create the file **full.html**.

The contest administrator can generate additional HTML output files by placing additional XSL files in the **data/xsl** folder; each XSL file in the **data/xsl** folder will cause a corresponding HTML output file to be generated.[50] There are numerous sample XSL files in the **`samps\web\xsl`** directory which can be copied into **data/xsl** (and then edited as desired). There are also sample XSL files in **`samps\web\xsl`** that create per-region HTML files and World Finals HTML files; see the file **`samps\web\xsl\README`** for more details.

The XML standings string which is transformed by each **.xsl** transformation file is a hierarchical XML document containing a variety of data fields which can be accessed by the XSL transformations and which can therefore be used to "customize" the resulting HTML output file.

For example, the XML standings string contains a root node named **/contestStandings**, below which is a node named **/standingsHeader**. The **/standingsHeader** node contains two fields, **@title** (holding the current Contest Title) and **@scoreboardMessage** (holding a message indicating whether the standings are for a "live" (private) or "freezable" (public) scoreboard.

Thus, the inclusion of a line in an XSL transformation file such as

```
<h2><xsl:value-of select="/contestStandings/standingsHeader/@title"/></h2>
```

will place the Contest Title at that position in the HTML output (using <h2> font), while the appearance of a line such as

```
<xsl:value-of select="/contestStandings/standingsHeader/@scoreboardMessage"/>
```

in the XSL transformation file will place the "scoreboard message" at that position in the HTML output. See the PC^2 wiki page at https://github.com/pc2ccs/pc2v9/wiki/Scoreboard-Standings-Data-Fields for a complete listing of all the contest standings XML attributes which are accessible to XSL Transformation files.

---

[49] In fact, it is the *presence* of a **.xsl** file in the **data/xsl** folder which *causes* the generation of a corresponding HTML output file.

[50] The Scoreboard module can also generate output files other than HTML from the XML standings. Specifically, it also looks for files in the **data/xsl** folder whose names end with **.json.xsl**, **.tsv.xsl**, **.csv.xsl**, and **.php.xsl**, and if found it will apply the XSL transformations in those files to the XML standings, producing corresponding output files (**.json**, **.tsv**, **.csv**, or **.php**).

10.9 No-GUI Mode

Beginning with PC^2 Version 9.5 the scoreboard can be started in "no GUI" mode by adding the argument "**`--nogui`**" to the **pc2board** startup command. A scoreboard started in no-GUI (also called "headless") mode does not provide a direct graphical display of the current standings – but it *does* continually generate all the specified scoreboard HTML files as described above, including automatically updating them as described above.

This can be useful in a situation where the scoreboard machine is being managed remotely from a terminal without a graphical display. It can also be useful to avoid having the "actual current standings" visible in the Scoreboard module display if the module is being run at a location where people should only be allowed to see the "public standings" which respect the "scoreboard freeze".

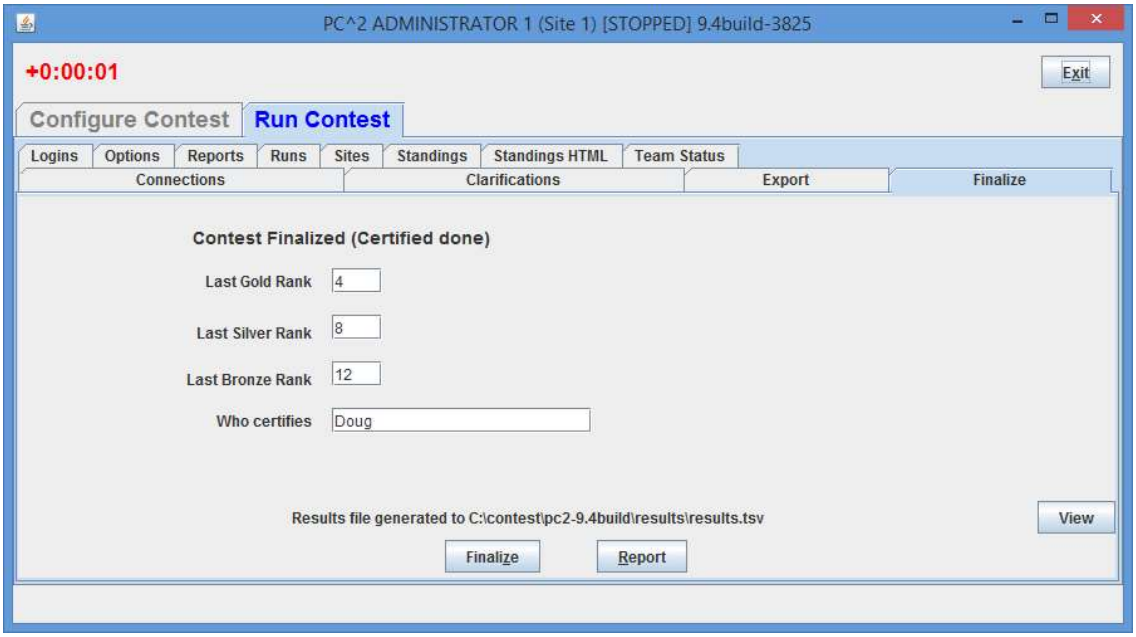
11 Finishing the Contest

When the contest is over, there are typically a number of additional steps required to "finish up". These include "finalizing" the results, optionally exporting the results for upload to another system, and shutting PC^2 down smoothly. This chapter discusses each of those tasks.

11.1 Finalizing

"Finalizing" a contest refers to certifying to PC^2 that the contest has been completed. Finalization checks several things: insuring that all submitted runs have been judged, that all clarification requests have been answered (even if just nominally), and that event feeds are closed properly. In addition, Finalization allows for providing an indication of what "medals" should be awarded. (This latter step is provided for compatibility with the CLICS CCS specification and is used at the ICPC World Finals; it may or may not be relevant to your own contest.)

Finalizing is done via the **Finalize** tab on the Admin **Run Contest** screen, shown below. Pressing the "**Finalize**" button causes PC^2 to check that all prerequisites for finalization have been met, and if so to mark the contest "finalized".



Finalizing a contest also has the side effect of generating a **results.tsv** file containing the final contest standings (this file can then be uploaded to external systems such as the ICPC Contest Management System; see below). Note that in the screen shot shown above, the "**Finalize**" button has already been pushed, as indicated by the message giving the location of the **results.tsv** file (this file will not be generated if the contest cannot be finalized for some reason, such as remaining unjudged runs). See the following section for more information on **results.tsv**.

Finalizing also has the additional effect of enabling the "public" scoreboard to be "Unfrozen" (see the section on **Options** in the chapter on **Interactive Contest Configuration**, as well as the chapter on the **PC² Scoreboard**, for additional details on scoreboard "unfreezing").

The **Report** button on the **Finalize** screen can be used to view and save a copy of a "Finalization Report" for the contest; the **View** button can be used to view and save a copy of the **results.tsv** file after it has been generated

11.2 Exporting Contest Results

When a contest is completed, it is sometimes desired to be able to transmit the contest results to other, external systems – for example, by uploading the results to the ICPC Contest Management System (CMS). PC^2 supports two different mechanisms for exporting contest results. The preferred method is to generate a **results.tsv** file. An older method, generating a **pc2export.dat** file, is also retained for backwards compatibility with older external systems.

11.2.1 Generating a *results.tsv* export file

PC^2 automatically generates a file named **results.tsv** when the contest is "Finalized" (see the section on Finalizing, above). The generated file contains a complete summary of contest results and matches the format for **results.tsv** specified by the CLICS CCS specification at https://clics.ecs.baylor.edu/index.php/Contest_Control_System#results.tsv; this file is therefore suitable for direct uploading to the ICPC CMS. See the Appendix on **Import/Export Interfaces** for further details on the format of the **`results.tsv`** file. The PC^2 Wiki page at https://pc2.ecs.csus.edu/wiki/Results.tsv also contains more information on the **results.tsv** file.

11.2.2 Generating a *pc2export.dat* export file

The PC^2 Scoreboard automatically generates a text file named **`pc2export.dat`** containing contest standings on the occurrence of any scoreboard refresh event as described earlier in the Scoreboard chapter. The **pc2export.dat** file is created in the directory where the Scoreboard was started.

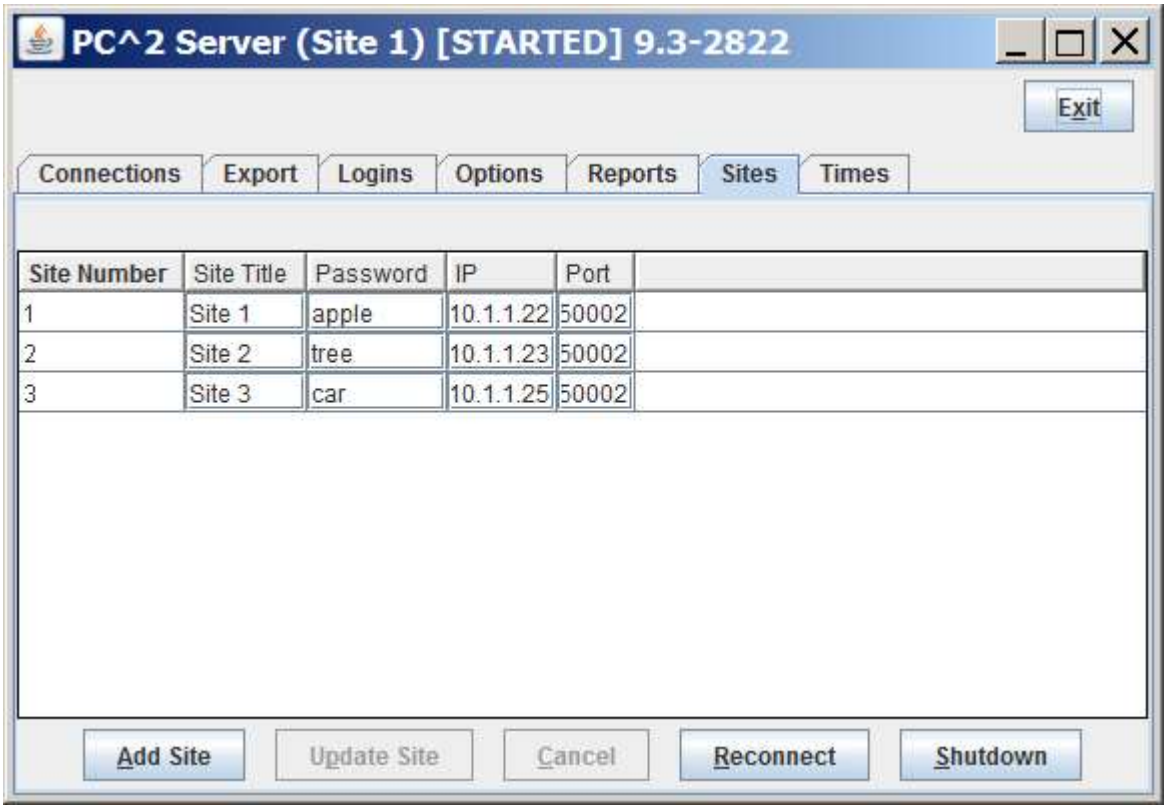
The **`pc2export.dat`** file contains, for each team, a record giving the number of problems solved, the total number of penalty points accrued on solved problems, and the time of last submission of a correct solution (used in the ICPC World Finals as a tiebreaker). See the Appendix on **Import/Export Interfaces** for further details on the format of the **`pc2export.dat`** file.

11.3 Shutting Down

Clients (team, judge, scoreboard, and admin) can be shut down by pressing the **Exit** button in the upper right corner of their GUI.

PC^2 servers can likewise be shut down by using the **Exit** button on the Server GUI (shown below). When a server is shutdown, ***all clients currently logged in to that server*** will also be shutdown.

For servers running in non-GUI mode, use the **Shutdown** button on the **Sites** tab on the Administrator: select a site and press **Shutdown** to shut down the server for that site. Selecting multiple sites and pressing Shutdown will shut down all those site servers.


PC^2 Server (Site 1) [STARTED] 9.3-2822
Exit
Connections
Export
Logins
Options
Reports
Sites
Times
Site Number
Site Title
Password
IP
Port
1
Site 1
apple
10.1.1.22
50002
2
Site 2
tree
10.1.1.23
50002
3
Site 3
car
10.1.1.25
50002
Add Site
Update Site
Cancel
Reconnect
Shutdown

Appendix A – pc2v9.ini Attributes

As described in the chapter on PC^2 Initialization Files, the **pc2v9.ini** file consists **[server]** and **[client]** sections, 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 **pc2v9.ini** file, along with a description of their function.

[server] section attributes:

port=<*portNumber*>

Tells the server the port number on which it should expect to be contacted by clients, and by other PC^2 servers in a multi-site contest. This attribute may be omitted from the **pc2v9.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)[51].

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

Tells the server the IP address and port number of a remote PC^2 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.

proxyme=true

Tells a server which is connecting to a remote server to ask that remote server to act as a proxy for communications with other servers. If this attribute is defined for a server, no other server will ever attempt to make an inbound connection to this server – meaning it is not necessary to open up any firewall ports for inbound access on this server. If this server is a primary server (i.e. does not contain a **remoteServer=** entry in its **pc2v9.ini** file and hence does not connect to a remote server), this attribute has no effect.

baseRunNumber=<positive_integer>

The run ID number to be assigned to the first run submission from teams at this site. RunIDs are assigned increasing sequential integers starting with this value (the default

[51] http://www.iana.org/assignments/port-numbers

baseRunNumber value is 1 *for each site* if this attribute is not present in that site's **pc2v9.ini** file). This attribute allows the contest administrator to guarantee that run IDs are unique across all sites , by assigning (for example) **baseRunNumber=1001** on site one, **baseRunNumber=2001** on site two, etc.

[client] section attributes:

server=<IPAddress>:<portNumber>

Specifies the IP address and port number at which the client module should contact the PC^2 server. Every client module MUST have a "**server=*<IPAddress>:<portNumber>***" entry in the **[client]** section of its **pc2v9.ini** file. The IP address and port number must correspond to the address of the machine running the PC^2 server and the port number at which the server on that machine is expecting to be contacted.

plaf=<type>

Specifies the "Programmable Look-And-Feel (PLAF)" which should be used in displaying the client GUI. Allowable values for ***<type>*** are "**java**", which causes the GUI to use the standard Java GUI appearance (which means that client GUIs will look the same regardless of the underlying platform), and "**native**", which causes the GUI to use the underlying platform's "native look-and-feel" – so for example on a Windows machine the GUI will look "Windows-like" while on a Mac the same GUI will look "Mac-like".

Appendix B – Networking Constraints

Overview

As mentioned in the beginning of this manual, PC^2 modules must be able to communicate with each other via TCP: clients must be able to communicate with their servers, and servers in a multi-site contest must be able to communicate (directly or via *proxies*) with other servers. If client machines reside on the same network 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^2. In order to understand how to avoid (or circumvent) these problems, it is useful to have some understanding of how PC^2 networking is implemented.

PC^2 is written in Java and uses TCP sockets for communication between modules. Server modules listen for incoming connections, using port 50002 by default.[52] Client machines initiate connections to a (single) server based on the **`server=IP:port`** attribute in the **`[client]`** section of the client machine's *pc2v9.ini* file. Client machines *only* communicate with the server specified in their *pc2v9.ini* file.

Server to Server Communication

When a server is started, it looks in the **`[server]`** section of its *pc2v9.ini* file. If there is no **`remoteServer=IP:port`** entry in the file, the server assumes it is the "primary" (first) server in the contest, and waits for inbound connections from clients and/or other servers.

If the **`[server]`** section of the *pc2v9.ini* file does contain a **`remoteServer=IP:port`** entry, then the server assumes it is a "secondary" server and it initiates an outbound connection to the remote server at the specified IP:port address. (This in turn implies that the primary server must always be configured to accept such an inbound connection; there must not be any firewall blocking inbound connections on the specified port at the primary server.)

Once a secondary server has connected to the primary server, it receives back information identifying any *other* servers which may also be connected to the contest. By default it then attempts to make a direct connection to those other servers. This in turn implies that by default all secondary servers must be configured to accept inbound connections from other servers – that is, there must not be any firewall blocking access to inbound connections on secondary servers. (See the section on Proxy Sites below for information on how to avoid the necessity of opening up inbound ports on secondary servers.)

[52] The listening port is configurable via the *pc2v9.ini* file.

Proxy Sites

In situations where it is desired to avoid having to open up firewall ports on secondary servers, it is possible to arrange for *proxying*[53]. When a secondary server is started, it looks in its *pc2v9.ini* file for an entry of the form **`proxyme=true`**. If this entry is present, then when the secondary server first contacts the primary server it requests that the primary should act as a proxy for all communications with any other servers.

When a secondary server has requested proxying like this, it will not attempt to make direct connections to any server other than the primary, and no other servers will ever attempt to directly contact that secondary server; all communications intended for the proxied secondary server will automatically be routed through the proxy. The effect of this is that secondary servers which are proxied will never be contacted by any server other than their primary – hence, **a proxied secondary server does not have to open up any firewall ports for inbound connections**.

NAT

Another constraint on networking has to do with NAT (Network Address Translation). For PC^2 to work using NAT you must configure port forwarding on your firewall, and configure the PC^2 site table with the public address/port for remote connections. Note that this only applies to non-proxied servers; proxied servers do not receive inbound connection requests and therefore are not affected by port forwarding under NAT.

[53] Proxying is only available in PC^2 Version 9.6 and above.

Appendix C – PC² Server Command Line Arguments

The command to start a server is:

```
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:

```
$ pc2server --help
Usage: Starter [--help] [-F filename] [--server] [--first]
        [--login <login>]  [--password <pass>]
        [--load <dir>|<file>]
        [--skipini] [--ini filename]
        [--contestpassword <pass>]  [--nogui]
```

As seen from the "usage" output, the pc2server command actually runs a program named "Starter" (with a **--server** option). The Starter program accepts the following command line options:

-F: specifies a text file with command line options, an alternate to specifying sensitive information on the command line. See the section **Using the -F option** for more details.

--server: indicates that this Starter is to run as a server, otherwise starts as a client.

--first : indicates that this server is a primary server and should not attempt to contact any other servers (ignores any **remoteServer=** attribute in **pc2v9.ini**)

--login : specifies the PC² login account name

--password : specifies the PC² password

--load: loads a contest configuration from either a directory in CLICS CDP format or from a specified YAML file.

--skipini: ignores the **pc2v9.ini** file

--ini: specifies an override ini filename

--contestpassword: on the first server only, specifies the contest password.

--nogui: starts this server without a graphical user interface. See the section **Non-GUI Server Startup** for more details.

Using the –F option

The –F command line option will load command line options from an input text file. This option is a security feature. Under most Unix systems the complete command line is listed when using a **ps** or similar command revealing login ids and passwords. Using the -F option, login ids and passwords can be stored in a text file. Note that the command line options are not limited to login and password options; any command line option can be stored in the specified text file.

If this command line was used:

```
pc2server --nogui --contestpassword cpass --login site1 --password site1pass
```

One could alternatively use the -F option:

```
pc2server -F secure.txt
```

where the secure.txt file contains

```
#
# Command line for non GUI server
#
--nogui
--contestpassword cpass
--login site1
--password site1pass
```

Blank lines and lines starting with # are ignored in the file (secure.txt).

The order that command line values are applied (highest precedent first) are:

1. specified -F option properties
2. specified on the command line
3. pc2v9.ini

Appendix D – ICPC Import/Export Interfaces

D.1 Importing ICPC Registration Data

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

To import ICPC registration data to PC^2, the RCD must first log into the ICPC CMS and download the "PC^2 Initialization" zip file which is automatically created and updated as changes in registration data occur.[54] Once the PC^2 Initialization data file is downloaded, it should be "unzipped" at any convenient location.

Unzipping the PC^2 Initialization data file will produce four 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; "**`PC2_Team.tab`**", containing data about the teams that are registered in the contest; and "**`_PC2_Team.tab`**". This last file contains the same data as in the **`PC2_Team.tab`** file but has an additional column which is initially filled with "null" and is intended to be filled in by the contest administrator in order to specify the PC^2 team number to be associated with each team.

Note that PC^2 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^2 accounts, the "**`_PC2_Team.tab`**" file can be edited by adding the PC^2 team number in the leftmost column. The specified team numbers will then be assigned to the corresponding teams when the initialization file is loaded into PC^2.

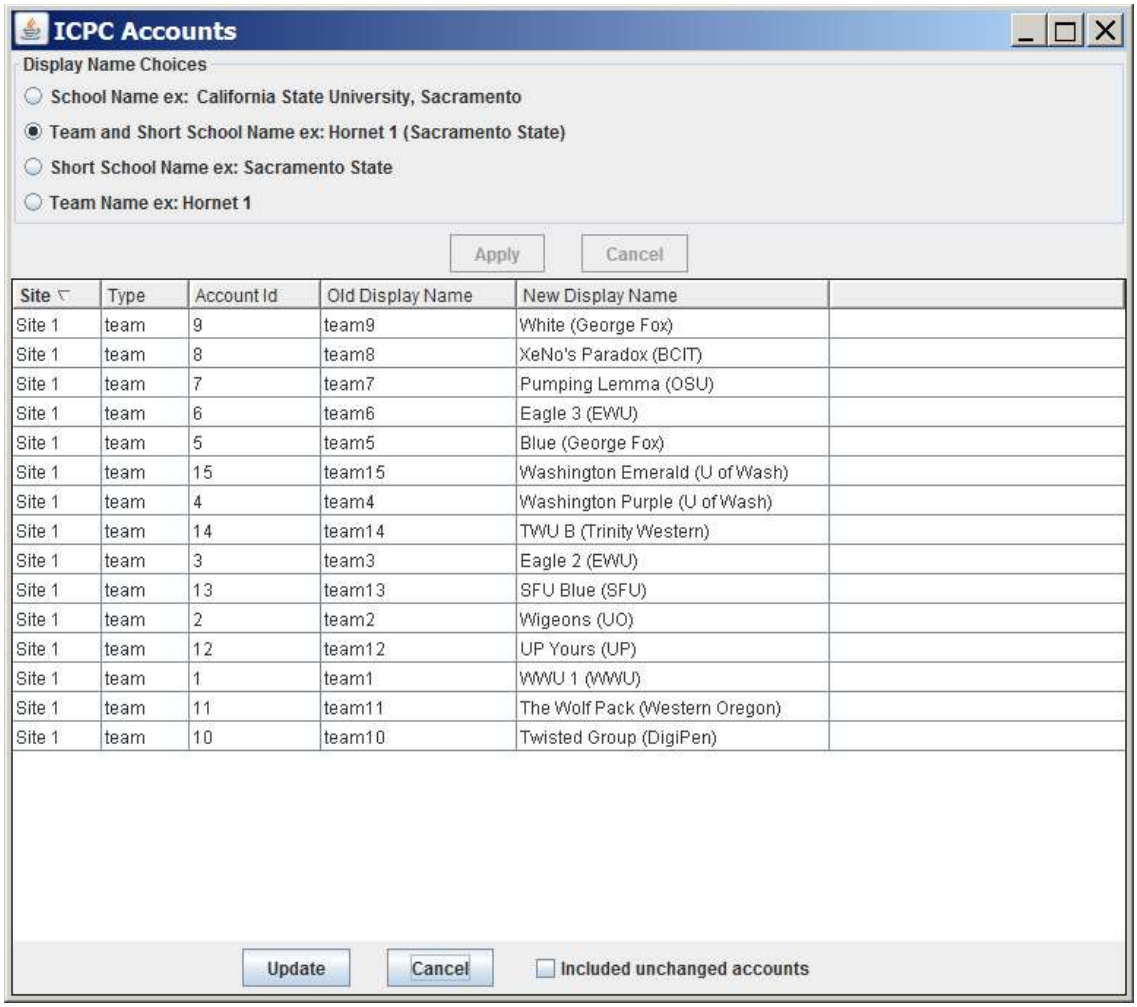
PC^2 expects quotation marks in the team data to be "quoted". That is, if any field in the data contains a quotation mark (**`"`**), then (1) the entire field must be surrounded by an additional matching set of quotation marks, and also (2) each quotation mark which is part of the data must be doubled. Thus for example a team name like **`The "TOPS" Team`** should appear in the import data file as **`"The ""TOPS"" Team"`**. If the PC^2 Initialization file exported from the ICPC CMS contains data with quotation marks, 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^2, you may see "format error" messages in the log file, and the data containing the quotes will not be displayed properly.

[54] The PC^2 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^2.

To load the ICPC import data into PC^2, select the **ICPC** tab on the **Configure Contest** tab on the main Administrator screen. This will produce the following screen:

The next step is to specify the sites which are to be imported into the system. Press the "Import Sites" button on screen; this will cause PC^2 to display a "file selection" dialog. Navigate to the location where the import files were unzipped and select the **PC2_Site.tab** file. This will load *both* the **PC2_Site.tab** file *and* the **PC2_Contest.tab** file. Next, press the "Import Accounts" button, navigate to the directory containing the import files, and select the **_PC2_Team.tab**. Note: all initialization ".tab" files must all reside in the *same directory*.

Selecting the PC2_Team.tab file will produce the **Change Display Format** screen, shown below (this screen can also be invoked by pressing the **Change Display Format** button after completing an account import). Each record in the ICPC PC^2 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^2 can be configured to use any one of these names, or a combination of them, as the name to be displayed on the scoreboard; that is the purpose of the **Change Display Format** screen.



Select the desired **`Display Name Choice`** on the **`Change Display Format`** screen and then press the **`Apply`** button. **`Apply`** changes the values displayed in the **`New Display Name`** column to match the chosen option.

When the **`Change Display Format`** screen is invoked it displays only information for accounts which have changed. During an initial import operation, all accounts will be displayed; however, if the screen is subsequently invoked by pressing the **`Change Display Format`** button it will initially be empty (since no accounts have changed). Click the "Include unchanged accounts" checkbox to display all accounts.

Once the desired account configuration is set up, click the **`Update`** button to save the new display names in the system.

D.2 Exporting Contest Results

As described earlier in this manual, PC2 supports two different file formats for exporting contest results: the newer **results.tsv** format and the older **pc2export.dat** format (retained for backward compatibility). This section describes the layouts of each of these files

D.2.1 The *results.tsv* File

The format of the **results.tsv** file is as defined in the CLICS CCS specification (see https://pc2.ecs.csus.edu/wiki/CLICS for details regarding CLICS). **Results.tsv** is text file consisting of a single "version line" followed by a series of lines, one for each team in the contest, sorted in rank order with alphabetical order on team name as tie breaker. Each line has *tab separated fields* as defined below.

The first line has the following format:

`results 1` ('1' represents the file format version number)

Then follow several lines (one per team) with tab-separated fields as follows:

`TeamID` – an integer; for uploads to ICPC CMS, this must be the team's CMS "external ID"

`Rank In Contest` – an integer giving the rank this team earned

`Award` – a string listing any award(s) the team one; e.g. "Gold Medal"

`Problems Solved` – an integer giving the number of problems the team solved

`TotalTime` – an integer giving the total "time penalty" the team accrued

`Time Of Last Submission` – an integer giving the contest time at which the team's last successful submission took place

`Group Winner` – a string listing any Group (Region) in which the team finished first (e.g., "North America")

The **results.tsv** file is automatically generated when the contest is "finalized" (see the section on Finalizing the Contest). It can also be generated manually by pushing the **Refresh** button on the PC^2 Scoreboard client.

D.2.2 The *pc2export.dat* File

Pressing the "**Refresh**" button on the scoreboard display causes the scoreboard to generate a new **`pc2export.dat`** text file containing the contest results in the old form required by the ICPC

CMS.[55] 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^2 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 **wf.standings** in the samps/web/xsl directory 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 Contest Management 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. 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^2 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 empty. 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^2** team account number in lieu of the ICPC Team ID. This will cause the "**Export ICPC**" operation to generate a file containing all the data necessary to compute complete contest standings utilizing PC^2 account numbers.

D.3 The *PC2_Team* File

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^2 Team ID; see above).
2) An integer giving the ICPC Region ID (the "region" or "group" to which the team belongs).

[55] Actually, the export data file is automatically generated (updated) each time the contest standings change; pressing the "Refresh" button simply displays a message showing its location.

3) A single character indicating the Team's "registration status" for the contest – typically either 'P' (pending), 'A' (accepted), or 'C' (cancelled). PC^2 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^2 ignores this field (but it must be present).

Appendix E – Output Validators

E.1 Overview

PC^2 allows the Contest Administrator to configure each problem so that it has associated with it a *validator* 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).[56] The validator program contains logic to make a determination, according to some set of rules, regarding the correctness of the team's output. A validator can also return the result of its determination to PC^2, making it possible to totally automate the judging process.[57]

A 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, it could perform "difference testing" between the team's program's output and a Judge's "answer file").

PC^2 utilizes a set of "interface conventions" defining both how information is passed from PC^2 to a validator and how the validator returns to PC^2 an indication of what judgment it thinks should be applied to the run. PC^2 can be configured either to accept the validator judgment as final (called "fully automated" or "computer" judging), or it can be configured such that the validator result is displayed to a human judge as a "recommendation" (in which case the human judge makes the determination of whether to accept the validator recommendation or instead to assign some other judgment to the team's submission).

E.2 Validator Selection

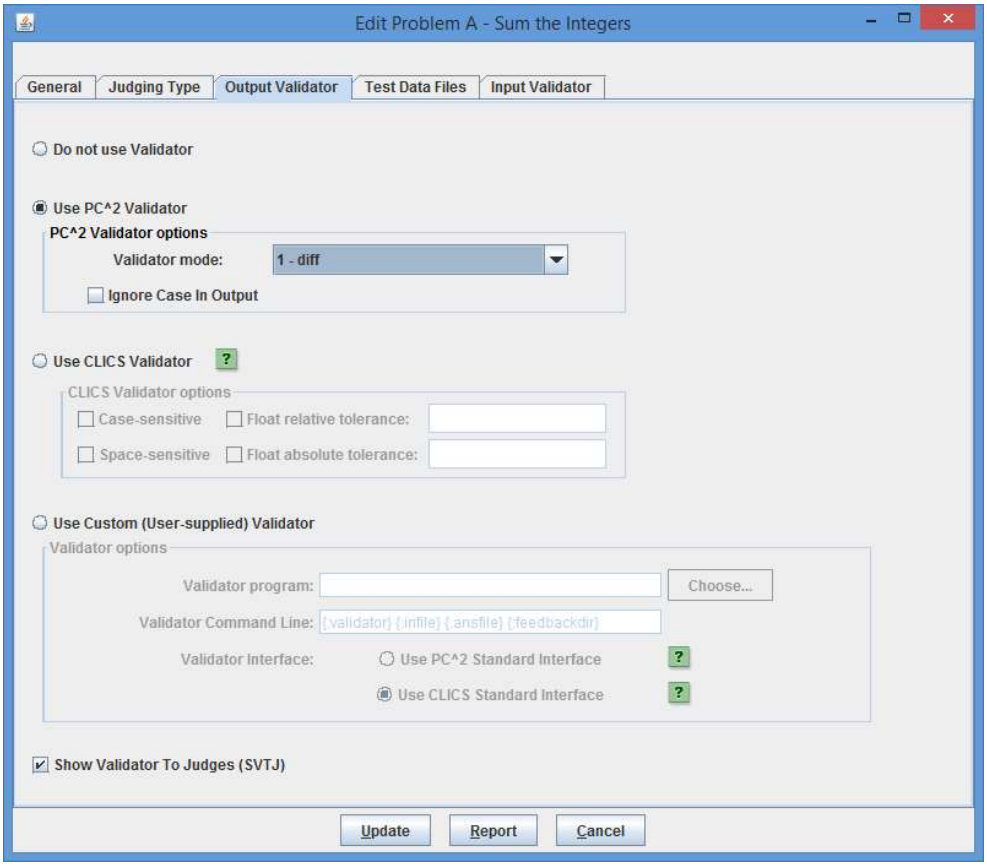
By default there is no validator attached to (associated with) a contest problem in PC^2. Validators can be attached to a problem by the Contest Administrator by using the **Output Validator** tab on the **Edit Problem** dialog. This displays the Validator Configuration screen, shown below. When the Contest Administrator configures a problem to use a validator, then when a judge (automated or human) executes a team program the specified validator will automatically be invoked as soon as the team program completes execution.

The Contest Administrator can choose one of three options for attaching an output validator to a contest problem: the simple "built-in PC^2 Validator"; an implementation of the more robust "CLICS Validator"; or a custom (user-supplied) validator.

---

[56] The CLICS CCS specification actually defines two types of validators: *input validators*, which examine the judge's input data to insure the data complies with the specifications given in the contest problem statement, and *output validators*, which examine the output of a team's program for correctness. PC^2 supports both types of validators. However, since output validators are much more common, the term "validator" by itself is generally interpreted as referring to *output* validators, and that is the how the stand-alone term is interpreted in PC^2 documentation. This Appendix is about output validators; see the separate Appendix on Input Validators for further information on that type of validator.

[57] See the section **Assigning Auto Judging to Judge modules** in the chapter on **Interactive Contest Configuration** for additional information on setting up automated judging.



The Contest Administrator can choose whether or not to display the validator result to the (human) 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 "Report" button pops up a window displaying the configuration of the current problem (not just the Validator portion), and allows saving the report to a file.

E.2.1 The Simple Built-in PC² Validator

The built-in PC² validator is essentially a simple version of the Unix "diff" program, with the ability to choose some options. The choices include:

1. Perform a straight "diff" between the team's output and the judge's answer file
2. Perform a diff, but ignore any whitespace at the beginning of the team's output
3. Perform a diff, but ignore any leading whitespace on lines of the team's output
4. Perform a diff, but ignore all whitespace on lines in the team's output
5. Ignore empty lines in the team's output

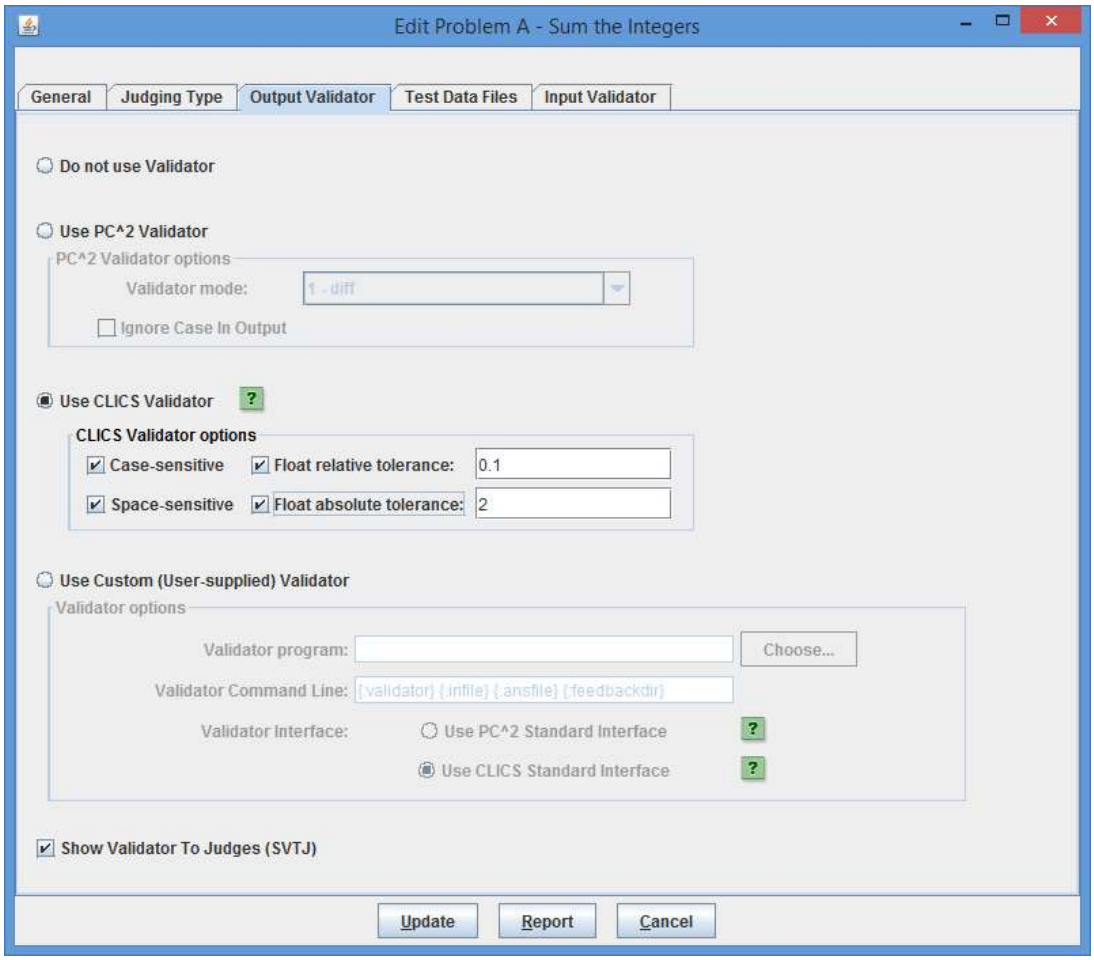
The above choices are mutually exclusive; there is no way to combine the options (see the term "simple", above). The simple PC² validator also separately supports the ability to specify

whether the validator should or should not ignore character case (upper vs. lower) in the team's output, by checking the corresponding box shown on the above screen.

E.2.2 The CLICS Validator

PC^2 supports an implementation of the default validator defined by the CLICS CCS specification, found at https://clics.ecs.baylor.edu/index.php/Problem_format#Output_Validators. This is the validator typically used at the ICPC World Finals.

The following screen shows the options available when the CLICS Validator implementation is selected:



Checking the "Case-sensitive" checkbox causes the PC^2 CLICS Validator to reject submissions which do not match the judge's answer in character case (if the box is unchecked, the validator will judge a submission as "correct" if the only differences are in character case).

Checking the "Space-sensitive" checkbox causes the validator to require that spacing in the team's output exactly match that of the judge's answer; if the box is unchecked then the validator essentially "tokenizes" the team's output, ignoring whitespace, and compares the resulting tokens with those found in the judge's answer file.

Checking the "Float relative tolerance" checkbox allows the Contest Administrator to specify a *relative tolerance* for floating point values in the team's output; if a floating point value in the team's output is within the specified relative tolerance of the corresponding value in the judge's answer, the team value is accepted as correct. The relative tolerance value is a percentage, represented as a decimal value between zero and one with "1" representing 100%; for example, specifying a float relative tolerance of 0.1 indicates that the team's output will be accepted if it is within 10% of the judge's answer.

Checking the "Float absolute tolerance" checkbox allows the Contest Administrator to specify an *absolute tolerance* for floating point values in the team's output; if a floating point value in the team's output is within the specified absolute tolerance of the corresponding value in the judge's answer, the team value is accepted as correct. For example, specifying a float absolute tolerance of 2 indicates that the team's output will be accepted if it is within 2 units of the judge's answer.

The interpretation of combining float absolute and relative tolerances is that if *both* are specified, then the team's output value is accepted as correct if it lies *either* within the specified float absolute tolerance *or* within the specified float relative tolerance. (This matches the CLICS Default Validator specification).

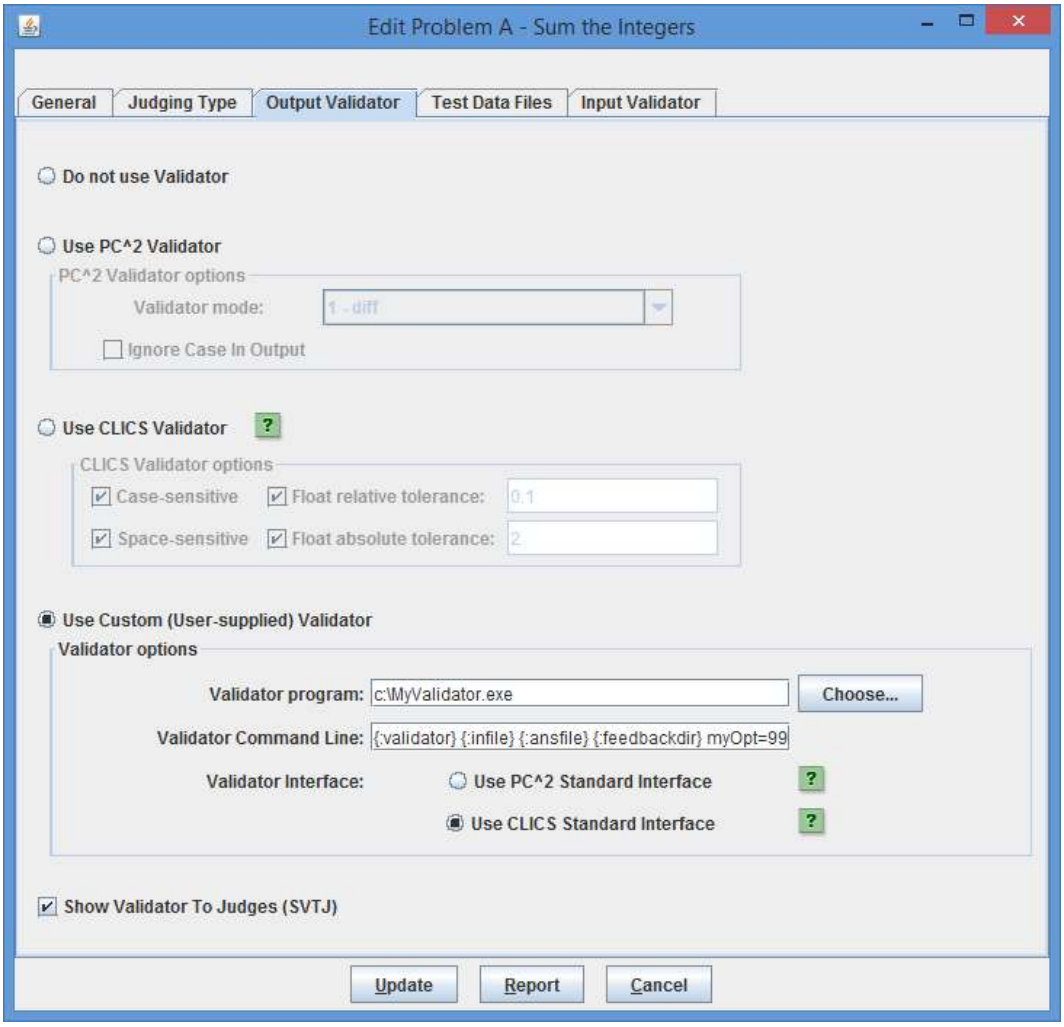
E.2.3 Custom (User-supplied) Validators

The third validator option is to use a separate external program as the validator.[58] In order to use this option the Contest Administrator must specify three things:

1. The full path to the external validator program file (the executable program, script, etc.)
2. The command line which is used to invoke the external validator program, and
3. An indication of the *interface convention* used by the external validator program (that is, the specification of how the validator interfaces with PC^2 both to get its input and to return its output).

The following screen shows a contest problem being configured to use a custom validator.

[58] Here, "program" is used in the general sense; a user-supplied custom validator could be a user-written script, an executable file generated by externally compiling a program, an executable JAR file, an invocation of an interpreter, or a combination of these. The only requirements are that the validator be invokable via a command line and that the validator conforms to one of the supported PC^2 "validator interface conventions" (see the following sections).



In the example shown above the validator program is an executable file named "**MyValidator.exe**" which has been written to use the "CLICS Validator Interface" convention and is invoked using the following command line:

```
{:validator} {:infile} {:ansfile} {:feedbackdir} myOpt=99
```

"**myOpt=99**" is an argument that the **MyValidator.exe** program is expected to accept and process (along with the substituted values of the other command parameters, described below).

The **Choose...** button can be used to navigate to and select the program to be used as the custom validator; it will automatically insert the full path to the selected program file into the **Validator program** textbox. If the program name is instead typed directly into the **Validator program** textbox, be sure to include the *full path to the program* as part of the program name. See the following sections for explanations of validator interfaces and of the fields in the above command.

E.3 Custom Validator Invocation Command Lines

The Contest Administrator enters the command to be used by PC^2 to invoke the validator into the **Validator Command Line** textbox. Following the execution of the team's program on each judge's data set configured in the contest problem, the system immediately invokes the specified validator command just as if the command had been typed at a console window on the host platform.

The Validator Command can include *parameter substitutions* similar to those allowed when configuring languages. As when defining contest 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 applied to the command line by PC^2 prior to invoking the validator:

Keyword	Meaning
`validator`	Represents the file name given in the **Validator Program** box.
`infile`	Represents the problem data input file as configured in the problem. (If the problem was configured with multiple input data files then this represents the specific data file used during the execution of the team program which produced the output to be validated.)
`outfile`	Represents the output sent to *stdout* by the team program when it was executed by the judge using the current input data file.
`ansfile`	Represents the judge's answer file corresponding to the current input data file, as configured in problem.
`resfile`	Specifies the name of the file into which a validator is expected to place an XML representation of the validator result judgment (used with PC^2 Interface Standard validators; see below).
`feedbackDir`	Specifies the name of a directory into which the validator may write feedback information (used with CLICS Interface Standard validators; see below).

Note that while the example in the previous section uses an existing program (the program **MyValidator.exe**) as the "validator program", it is not a requirement that the "validator program" actually be an executable program, nor is it a requirement that the item listed in the **Validator program** field actually be what gets invoked via the **Validator Command Line**. 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.[59]

[59] It is arguable that a better name for the label "**Validator Program**" would have been "**Validator File**", since it does not have to be a program…

For example, suppose a contest problem requires 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.)

One special case applies to the invocation of validators: if the name of the Validator program ends with "**`.jar`**", PC^2 assumes it represents a "runnable JAR file" – that is, a JAR file containing a Manifest entry listing the main method in the JAR – and the system automatically prepends "**`java -jar`** " to the front of the command, causing the JAR file to be properly executed. In all other cases the system simply invokes the specified Validator program as if the Validator Command Line had been typed at a console.

E.4 Custom Validator Interfaces

In order to support the use of custom validators in a Contest Control System (CCS), it is necessary to define a set of conventions for interfacing such validators to the CCS. This requires two things: a standard mechanism for passing data from the CCS to the validator, and a standard mechanism for passing validator results back to the CCS. Because a validator is a problem-specific entity (not a 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^2.

There are two such validator interface standards in common use today.[60] The first is referred to as the *PC^2 Validator Interface Standard*.[61] This standard was developed under the auspices of the ICPC by the PC^2 development team, working in conjunction with a number of other Contest Control System development teams. This standard is supported by PC^2 (going all the way back to Version 7) as well as by other Contest Control Systems.

The second, newer validator interface standard was defined under the auspices of CLICS (see https://clics.ecs.baylor.edu/index.php/Problem_format#Validators on the CLICS Wiki as well as https://pc2.ecs.csus.edu/wiki/Validator#CLICS_Output_Validator on the PC^2 Wiki).

The current version of PC^2 supports both the older version of the validator interface standard as well as the newer CLICS validator standard. Both standards specify two interfaces: the interface used by the contest control system to invoke the validator program, and the interface used by the validator program to pass results back to the contest control system. When the Contest Administrator selects a custom validator, the radio buttons on the Validator Configuration screen (above) must be

[60] As someone once said, "Standards are a good thing – we should have lots of them." Case in point.

[61] http://www.ecs.csus.edu/pc2/doc/valistandard.html

used to indicate which standard the custom validator uses (and obviously, the custom validator code must be written so that it conforms to the specified standard).

The following sections give a description of the interfaces defined by the two standards. Since PC^2 supports both standards, any validator written to comply with either of these interfaces can be used as a custom validator for a problem in PC^2.

One important thing to be aware of when creating a custom validator is that, regardless of which validator interface is used, PC^2 always arranges that the standard output and standard error channels of the validator are captured and made available to the judge following the execution/validation of a team's program. This can be particularly useful when trying to debug a custom validator, or when trying to find system configuration errors which produce unexpected validation results. For example, if a custom validator writes an error message to standard error when it receives incorrect or insufficient parameters, it will be easy to see this by looking at the standard error channel results on the judge. This in turn might lead to making a correction in the Validator Command Line configured for the problem. See the separate *PC^2 V9 Judge's Guide* for information on how to examine the Validator standard output and standard error output.

E.4.1 Using the PC^2 Validator Interface

E.4.1.1 PC^2 Validator Input Interface

The PC^2 Validator Interface specifies that the contest control system is responsible for passing at least four command line arguments to the validator, as follows:

argument1: a string specifying the name of the input data file which was used to test the program whose results are being validated.

argument2: 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').

argument3: 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 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.

argument4: 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 arguments to a validator can be met by the Contest Administrator in PC^2 through the use respectively of the **{:infile}**, **{:outfile}**, **{:ansfile}**, and **{:resfile}** command substitution parameters in the **Validator Command Line**; PC^2 will automatically insert the

appropriate values for the problem when the validator is invoked. Also, as required by the standard, PC^2 arranges that 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 a custom user-written validator program should expect to be invoked by PC^2 when using the PC^2 Validator Interface Standard.

The PC^2 Validator Interface standard also 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 further specifies that the interpretation of any such parameters is up to the validator. The Contest Administrator in PC^2 can pass arbitrary additional parameters to the validator by including them in the **Validator Command Line** after the four required parameters. An example is shown in the previous section, where the additional parameters "**`MyOption=99`**" is passed to the validator (this is a specific example; any set of additional arguments can be passed on the command line as long as they are expected by the validator program).

E.4.1.2 PC^2 Validator Result Interface

The PC^2 Validator Interface 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[62], 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. The standard specifies that if the value of "**string1**" is "accepted" (or any case-variation of that word), 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).

In PC^2, the appearance of any form of the word "accepted" in the "**string1**" attribute of the result element in the result file causes PC^2 to assign a recommendation of "YES" to the run being executed. In addition, any form of the word "Yes" also causes PC^2 to assign a recommendation of "YES" to the run being executed (this is an extension to the standard). Recommendations are displayed to the judge if the SVTJ checkbox has been checked when the validator is configured with the contest problem.

---

[62] 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^2 implementation expects a validator result file to have an XML header.

If the value of "**string1**" returned by the validator is not some form of the word "accepted" or "yes", then PC^2 compares the actual string value to the set of "judgment messages" currently defined in the system. If "**string1**" matches one of the currently-defined judgment messages, then PC^2 assigns that message as the recommendation for the run being executed; otherwise, it assigns a recommendation of "Undetermined" to the run.

The set of judgment messages recognized by PC^2 is defined on the **Judgments** tab of the Configure Contest tab on the main Administrator screen. The default set of judgment messages is given in the following list:

```
Yes
No - Compilation Error
No - Run-time Error
No - Time-limit Exceeded
No - Wrong Answer
No - Excessive Output
No - Output Format Error
No - Other - Contact Staff
```

See the section on **Contest Judgments** for information on adding to or editing the existing judgment messages.

"**string2**" in the XML file returned by a validator 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^2 does not use the "**string2**" parameter from the result file.

E.4.1.3 PC^2 Extensions

The PC^2 Validator Interface 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^2 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^2 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^2 invokes a validator it generates a unique random name for the result file. When the validator returns, PC^2 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^2 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.

E.5.2 Using the CLICS Validator Interface

E.5.2.1 CLICS Validator Input Interface

The CLICS Validator Interface standard specifies that the Contest Control System will invoke the validator by passing it at least three command line arguments, in order:

argument1: a string specifying the name of the input data file which was used to test the program whose results are being validated.

argument2: 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 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 input file 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. The meaning of the contents of the answer file is not defined by the standard.

argument3: a string which specifies the name of a "feedback directory" in which the validator can produce "feedback files" in order to report additional information on the validation of the output file. The feedbackdir must end with a path separator (typically '/' or '\' depending on operating system), so that simply appending a filename to **feedbackdir** gives the path to a file in the feedback directory.

In addition, the standard specifies that the team's output (the output file which was produced by the program being validated) must be presented to the validator's standard input channel; PC^2 arranges that this is true for any custom validator which is specified as using the CLICS Validator Interface standard.

The standard also specifies that the two files named by the **input** and **judge answer** arguments (arguments 1 and 2) must exist (though they are allowed to be empty) and that the validator program must be allowed to open them for reading. The directory pointed to by **feedbackdir** (argument 3) must also exist.

A sample **Validator Command Line** for invoking a custom validator using a CLICs interface in PC^2 might therefore look like the following:

```
{:validator} {:infile} {:ansfile} {:feedbackdir}
```

This command line will invoke the specified Validator program, passing it the current input data file name, the current judge's answer file name, and the name of a feedback directory. Note that when PC^2 invokes a custom validator using the CLICS Validator interface, the system automatically creates a new, randomly-named directory for use as the "feedback directory". This is done as a security measure; it prevents a team submission from using *a priori* knowledge about where "validator feedback" is to be written and attempting to somehow overwrite it. The Validator

program should use the specified feedback directory name as the directory where it writes any feedback files (see the following section for information on feedback files).

There is nothing in the CLICS Validator standard which prohibits *additional* command line arguments being passed to a CLICS validator. For example, a custom validator could be written to examine the command line for additional parameters (following the **feedbackDir** (3rd) argument) such as "**`case-sensitive`**", "**`space-sensitive`**", "**`float-tolerance`**", etc., and use these additional parameters to direct the validation process. Any such additional parameters would be added to the **Validator Command Line** specified in the **Use Custom Validator** portion of the Validator Configuration screen; so for example the **Validator Command Line** might instead look like:

```
{:validator} {:infile} {:ansfile} {:feedbackdir} case-sensitive
```

The interpretation of any additional parameters, as with the interpretation of the required parameters, is up to the code comprising the custom validator.

E.5.2.2 CLICS Validator Result Interface

The CLICS Validator Interface standard specifies that a validator program is required to report its judgment by exiting with specific exit codes:

- If the output is a correct output for the input file (i.e., the submission that produced the output is to be Accepted), the validator exits with exit code 42.
- If the output is incorrect (i.e., the submission that produced the output is to be judged as Wrong Answer), the validator exits with exit code 43.

The standard also specifies that "*Any other exit code (including 0!) indicates that the validator did not operate properly, and the contest control system invoking the validator must take measures to report this to contest personnel.*"

PC2 supports this interface for custom validators configured to use the CLICS Standard. That is, it expects the custom validator to exit with the specified exit codes based on the validation results, and uses the exit codes to determine what judgement to apply to the submission.

The CLICS validator interface standard also specifies that the validator may (but is not required to) report more information back to the CCS than just the accept/reject verdict implied by the exit code. In particular, the validator may write arbitrary files into the feedback directory for purposes of passing additional information to the CCS.

PC2 supports the ability of a custom validator to provide additional feedback information by examining the feedback directory for two files. If it finds a file whose name is "**judgement.txt**", it assumes that this file contains a line describing the judgement to be applied to the submission. For example, a **judgement.txt** file might contain a string such as "accepted" or "yes", or instead might contain a string such as "wrong answer" or "output format error" or "spacing error". This judgement information can be used by the judge to determine why a submission was rejected by the validator.

PC^2 arranges that the contents of the **judgement.txt file** (if any) is displayed as the "Validator Recommends" answer. Note however that since the contents of the **judgement.txt** file are arbitrary and are defined by the external validator, there is no way for PC^2 to automatically know whether the contents of a given **judgement.txt** file represents a successful or a rejected run; it uses the validator exit code to determine this.

PC^2 also supports the ability of a custom validator to provide more detailed feedback information by examining the feedback directory for a file whose name is "**judgementdetails.txt**". If it finds a file matching this name, it reads the contents and also displays it for the judge. For example, a **judgementdetails.txt** file might contain a string such as "Mismatch at line 8: judge token was 5.5, team token was 4.9". This kind of additional information can be used by the judge to further determine why a submission was rejected by the validator.

Appendix F – Language Definitions

As described earlier in this manual, PC^2 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^2 *does* with a language definition. Language definitions are used by PC^2 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^2 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**"[63] 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^2 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^2 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.

[63] The term "execute directory" refers to the directory which is current when a submission is executed and in which all execution operations (e.g. compiling, linking, execution) occur. In PC^2 Version 9 the execute directory name is based on the logged in user. For example, the execute directory for Team 3 at Site 1 is named "executesite1team3"; the execute directory for Judge 3 at Site 2 is named "executesite2judge3", etc.

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^2 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^2 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}
Java
```

Language: GNU C++

```
GNU C++ (Unix / Windows)
g++ -lm -o {:basename}.exe {:mainfile}
{:basename}.exe
.\{:basename}.exe
GNU C++
```

Language: GNU C

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

Language: Perl

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

Language: Microsoft C++

```
Microsoft C++
cl.exe {:mainfile}
{:basename}.exe
.\{:basename}.exe
Microsoft C++
```

Language: Kylix Delphi

```
Kylix Delphi
dcc {:mainfile}
{:basename}
.\{:basename}
Kylix Delphi
```

Language: Kylix C++

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

Language: Free Pascal

```
Free Pascal
fpc {:mainfile}
{:basename}
.\{:basename}
Free Pascal
```

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^2 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^2 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^2 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^2. 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^2 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^2 :

```
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^2 . 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^2 language processing algorithm described above. This organization of language processing gives a great deal of flexibility to the Contest Administrator.

Appendix G – Using the PC² API

While the client interfaces (Admin, Judge, Team, and Board) in PC² 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, PC² provides a mechanism for users to create their own "custom clients" which interface with the rest of the PC² system.

The PC² API Java doc is in the distribution under `doc/api/index.html`. In the API Java doc there are code snippets which show how to use the API.

To use the API you must add the pc2.jar in the CLASSPATH (or build path).

Use the ServerConnection class to connect to the PC² server. Here is the code snippet from the ServerConnection Java doc that shows how to connect to the server and access the contest data.

```
String login = "team4";
String password = "team4";
try {
    ServerConnection serverConnection = new ServerConnection();
    IContest contest = serverConnection.login(login, password);
    //... code here to invoke methods in "contest";
    serverConnection.logoff();
} catch (LoginFailureException e) {
    System.out.println("Could not login because " + e.getMessage());
}
```

After a successful connection, the IContest instance can be used to access the contest data/information.

Appendix H – Troubleshooting / Getting Help

Before getting help from the PC^2 Team

There are a number of documents and references that contain information about using PC2, take the time and search these references before sending an email to the PC2 team.

1. Search this document for an answer
2. Search the PC^2 Wiki or use Google to search for answers

   http://pc2.ecs.csus.edu/wiki/Main_Page
3. Search the on-line FAQ

   http://pc2.ecs.csus.edu/doc/faq/
4. Search PC^2 Bugzilla

   http://pc2.ecs.csus.edu/bugzilla/

Getting help from the PC^2 Team

If you can not find an answer to your question, send the PC^2 team an email at pc2@ecs.csus.edu.

If you have attempted to use the PC^2 system and are having a problem, please email us a **pc2zip** file. This file is created by using the **pc2zip** script to create a special .zip file in the archive directory.

Appendix I – PC² Distribution Contents

This following tables describe the contents of a PC² software distribution. Each distribution contains a single (base) directory which contains these directories and files.

Directories

Directory Name	Contains
bin	scripts to start PC² modules
data	XSL Stylesheets and other data files
doc	API and user documentation
lib	PC² Java library (jars)
projects	PC²-related projects (e.g. WTI web team)
samps	sample files
samps/contests	sample contests, with YAML descriptions
samps/data/xsl	XSL descriptions for PC² scoreboards et.al.
samps/scripts	compile and other scripts
samps/src	Samples in C, C++, Java, etc.
samps/web	web resources and scripts
samps/web/xsl	samples for group XSL for HTML

Files

Filename	Description
README	Late breaking and important info
VERSION	Version information
pc2v9.ini	Example PC² V9 initialization file
pc2.ico	Default icon for PC² modules

Appendix J – Log files

Log files are stored under the **`logs/`** directory. In Version 9.3 and above, server log files are also stored under **`profiles/<ProfileID>/logs`**.

There are 4 different log file types:

1. startup log files – logging information before a module is logged in
2. module log files – logging information for a logged in module/client, **typically these are the files to check for errors when running PC²**
3. evaluations/judgments log – on server only, one line per judgment, see http://pc2.ecs.csus.edu/wiki/Evals.log for more details.
4. security log files – logging security issues when they happen

Appendix K – Reports Program

The **pc2report** program can be used to produce stand-alone reports about the state of the system. This program must be run on the PC^2 server machine (i.e., the machine on which the pc2server program is run). Each report generated by **pc2report** is identical in output content and form to the reports created using the Admin Report Tab.

The following examples use the contest password 'newpass'; replace 'newpass' with the contest password entered when the PC^2 server was initially started. [64]

Show Fastest Solution Summary report

```
$ pc2report --contestPassword newpass  'Fastest Solution Summary'
```

Show Runs report

```
$ pc2report --contestPassword newpass Runs
```

Show Runs report, use the report number (15) instead of spelling out report name

```
$ pc2report --contestPassword newpass  15
```

Usage

```
Usage: [options] reportName|## [[reportName|##][...]]

--profile name - profile name, default uses current profile.  name may be a ##
from --listp listing
--contestPassword padd  - password needed to decrypt pc2 data
--list         - list names of reports (and the report numbers)
--dir name     - alternate base directory name, by default uses profile dir
name
--site ##      - specify the site number
--listp        - list all profile names with numbers
--noProfile - do not use profile directory use pre version 9.2 location

reportName - name of report to print (or report number)
##         - number of report to print (numbers found using --list)

$ pc2reports --listp
1 - Id: Contest-1526060434834405723 description: Real Contest name: Contest
2 - Id: Contest 3--613094433664018852 description: Real Contest 3 name: Contest
3

Default name  : Contest
  Profile ID  : Contest-1526060434834405723
  Description : Real Contest
  Path        : profiles\Pdf812e23-4234-46ee-ad3c-4011c8cb885e
```

[64] See the earlier note regarding the use of the **–F** option to avoid putting plain-text passwords on the command line.

Each of these will print the same report:

```
$ pc2report --contestPassword newpass --profile Contest 3--613094433664018852
'Fastest Solution Summary'
$ pc2report --contestPassword newpass --profile 2 'Fastest Solution Summary'
$ pc2report --contestPassword newpass --profile Contest 3--613094433664018852 9
$ pc2report --contestPassword newpass --profile 2 9

Precedence for directory: --dir, --profile, then default profile dir

Version 9.3 20140802 (Saturday, August 2nd 2014 20:46 UTC) Java ver 1.7.0_55
build 2822 Windows 7 6.1 (x86)
```

List all reports available

```
$ pc2report --list

Report 1 Accounts
Report 2 Balloons Summary
Report 3 All Reports
Report 4 Contest Settings
Report 5 Contest XML
Report 6 Contest Analysis
Report 7 Solutions By Problem
Report 8 Submissions by Language
Report 9 Fastest Solutions Summary
Report 10 Fastest Solutions Per Problem
Report 11 Standings XML
Report 12 Logins
Report 13 Profiles
Report 14 Plugins
Report 15 Runs
Report 16 Clarifications
Report 17 Problems
Report 18 Languages
Report 19 Judgements
Report 20 Runs grouped by team
Report 21 Notification Settings
Report 22 Client Settings
Report 23 Groups
Report 24 Evaluations
Report 25 Runs (Version 8 content and format)
Report 26 Run 5 field
Report 27 Account Permissions Report
Report 28 Balloons Delivery
Report 29 Extract Replay Runs
Report 30 Run Notifications Sent
Report 31 Judgement Notifications
Report 32 Active Profile Clone Settings
Report 33 Sites
Report 34 Unused 2011 Event Feed XML
Report 35 Notifications XML
Report 36 Finalize-Certify
Report 37 Internal Dump
Report 38 Passwords
Report 39 accounts.tsv (team and judges)
Report 40 accounts.tsv (all accounts)
```

```
Report 41 runs.tsv Report
Report 42 JSON Standings
Report 43 Unused 2013 Event Feed XML
Report 44 userdata.tsv
Report 45 groups.tsv
Report 46 teams.tsv
Report 47 scoreboard.tsv
Report 48 submissions.tsv
Report 49 ICPC Tools Event Feed
Report 50 Auto Judging Settings
Report 51 Judging Analysis
Report 52 JSON 2016 Scoreboard
Report 53 Contest Data Package
```

Appendix L – PC² XML (Legacy) Event Feed

PC² is capable of generating *event feeds* which conform to various specifications managed under the auspices of the *Competitive Learning Initiative* (the so-called *CLI Contest System [CLICS] specifications*, which can be found at https://clics.ecs.baylor.edu/index.php/Main_Page). Included in CLICS (at https://clics.ecs.baylor.edu/index.php?title=Event_Feed_2016) is a specification for an XML-based event feed containing elements describing events which occur in a contest, including such things as configuration information (e.g., teams, problems, and languages), run submissions, judging results, etc.[65]

The XML event feed generated by PC² is compatible with the Event Feed described in the CLICS specification; refer to that specification for information on the content and structure of an XML Event Feed. This appendix describes how to access a PC² XML Event Feed.

PC² supports two types of XML Event Feeds (EFs): *static* and *dynamic*. A *static* XML EF is a text "snapshot" of the *current state* of contest events, in XML format. Static event feeds can be generated at any time during a contest (including after the contest is over) and contain all the information and in exactly the format described in the CLICS specification, but only for events that have already happened at the time the snapshot EF is created.

Dynamic event feeds are streams which can be connected to by an external tool and which provide continual updating as new events occur in the contest. The type of EF described in the CLICS specification corresponds to a PC² dynamic event feed (the CLICS specification makes no mention of static event feeds; these are an extension supported by PC²). The XML contents of a PC² static event feed from a snapshot taken after a contest is over and has been "finalized" (see below) will be exactly the same as the sequence of XML elements found in a PC² dynamic event feed at the end of the contest after the contest has been "finalized".

Static event feeds (that is, text containing the XML event feed elements) are created using the PC² "Reports" facility on the Admin. Selecting the "Reports" tab on the Admin Main Screen allows selection of a Report titled "Event Feed XML".[66] Viewing this Report will show the text of the current event feed output (that is, all the XML events which will have been sent out to any external tool listening to the dynamic event feed). The contents of the on-screen report can be copy/pasted into an external editor and then saved as a file.[67] Alternatively, pressing the "Save XML" button on the XML Event Feed Report screen will prompt for a file name and save the entire XML event feed (minus the PC² header and trailer text, but only the XML for those events which have already happened) in a file.

Accessing the PC² *dynamic* event feed is a bit more complicated – but has the significant advantage that an external tool can receive automatic event updates without further intervention by

[65] Note that the XML event feed is deprecated by CLICS, in favor of a newer JSON-based event feed which is integrated into the CLICS "Contest API". PC² supports both the older XML event feed (now referred to as the *Legacy Event* Feed), as well as the newer JSON event feed (see the *Web Services* appendix for information on the Contest API and the JSON event feed).

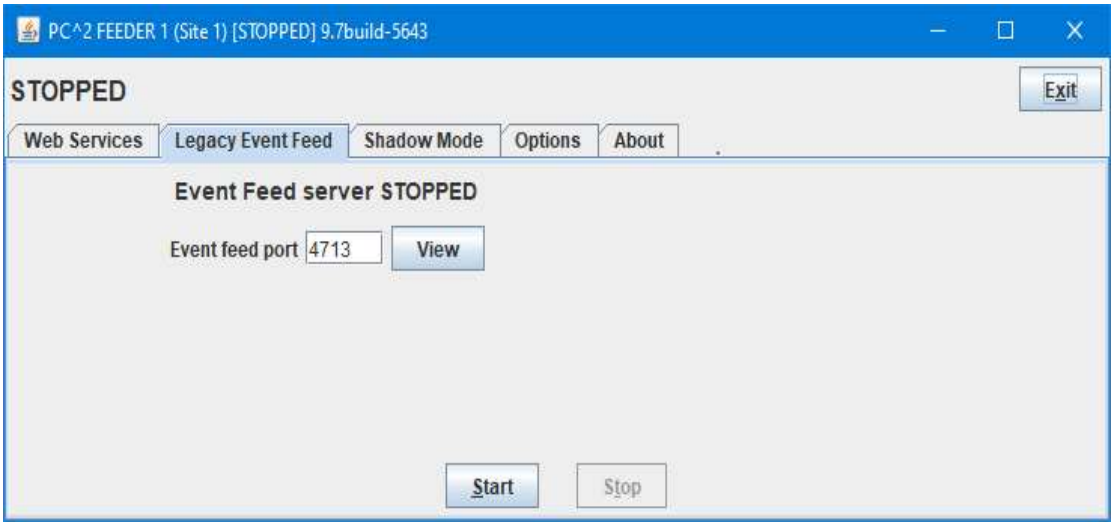
[66] In some versions of PC² this report may be titled "ICPC Tools Event Feed".

[67] When using this method be sure to omit the non-XML "header" text at the top of the report and the non-XML "trailer text" at the bottom of the report in order to obtain valid XML.

the Contest Administrator once the Event Feed is set up properly. PC^2 does not generate a dynamic event feed by default; in order to get it to do so it is necessary to start a special client called the "Event Feed (EF) Client", and then to start the EF Client listening for connections and outputting XML events.

To start the EF Client it is necessary to create an Event Feed Account (an account of type "FEEDER") and login to that account. Event Feed accounts are generated like any other PC^2 account: by selecting the Accounts tab on the Admin Configure Contest Screen, pressing "Generate" to display the account generation screen, entering the number of Event Feed accounts desired (normally there is no need for more than 1) and pressing "Generate Accounts" (see the section on Account Generation for additional details).

Once the Event Feed account is generated, use the script "**pc2ef**" (located in the **bin** folder along with all other PC^2 support scripts) to start a client and login with the "feeder" account and password (these default to "feeder1" and "feeder1" respectively, although this can be changed via the Admin Accounts tab). Logging in with a feeder account displays the Event Feed Server interface, shown below.



To start the Event Feed, select the desired port on which the Event Feed should be output (or accept the CLICS default port of 4713) and click the "Start" button; this starts the Event Feed server listening on the chosen port for connections and outputting XML events to that port as events occur during the contest. The above screen shows an Event Feed Server that is ready to be started listening on port 4713 with 4 hours and 57 minutes currently remaining in the contest. The "View" button will generate a text display of the current (static) event feed XML, without the extra PC^2 header/trailer information present in static event feeds generated using the Reports mechanism.

Note: *it is important to leave the Event Feed running for the entire contest*; stopping it will terminate the dynamic Event Feed. Restarting the Event Feed will *reinitialize it*, causing it to send all events from the beginning of the contest again.

The **pc2ef** command supports an optional "no GUI" capability. To start the Event Feed client with no GUI, use the command

```
pc2ef --nogui --login <account> --password <pw>
```

This command causes the Event Feed to be automatically started with no Graphical User Interface, using the specified account and password, and using the default port (4713). If the option "--password" is omitted then a default password for the specified account is attempted. Note that the options can be placed in a file using the –F option (see the section on using the –F option in the Appendix on PC^2 Server command line options for additional information).

See the Web Services Appendix for further details regarding the effects of starting the Event Feed client in **NOGUI** mode.

One important additional note should be mentioned. The CLICS specifications requires that a contest (and an Event Feed) be "finalized" after the contest is over. This process involves sending a <finalized> XML element to the Event Feed(s), which is the indicator to external tools that the contest is over. PC^2 supports the "finalize" operation via the "Finalize" tab on the "Run Contest" tab of the Admin; see the section on Finalizing in the chapter on Finishing the Contest for additional details.

The finalization process includes the specification of the ranks (team places) which should receive Gold, Silver, and Bronze medals in the contest (this is based on the ICPC World Finals, where Gold Medals are given to the top four teams, Silver Medals are given to the teams placing 5th through 8th, and Bronze Medals are given to teams placing 9th through 12th). These medal rank values are output as part of the <finalized> element in the Event Feed (although external tools are of course free to ignore them).

Note also that the output of the dynamic event feed is not "well-formatted XML" while the contest is running: the opening **<contest>** element will not have a corresponding **</contest>** until the contest is finished ("finalized").

One additional important note applies if you are running a *multi-site* contest (meaning, you are running multiple PC^2 Servers, as described in the section on Server Startup) and also the XML Event Feed is to be used as input to an external tool (for example, a tool such as the "ICPCTools Resolver"; see https://icpc.baylor.edu/icpctools/). The PC^2 XML Event Feed will automatically include information for teams at all sites (regardless of the site at which the Event Feed is generated). However, *it is important to insure that all teams in the contest have unique team numbers*. This is because the XML Event Feed outputs the team ID (team number), and does not distinguish between teams at different sites (the CLICS specifications were written without regard for the ability to run a multi-site contest as supported in PC^2). By default, team numbers at each site in a PC^2 multi-site contest start with "1" (so, there is a "team1" at Site 1, a "team1" at Site 2, and so forth). This is incompatible with the current ICPCTools Resolver (and possibly with other tools as well).

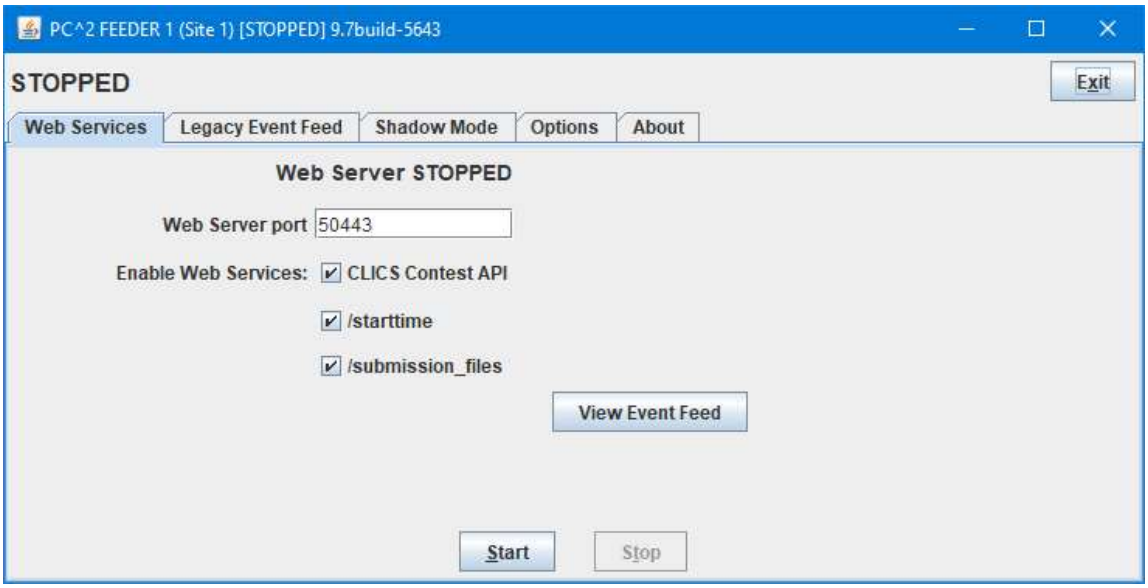
To avoid this problem, when using the PC^2 Admin "Generate Accounts" function to generate accounts for each site, be sure to use the "Start Account Number At…" field on the "Generate Accounts" pane to specify a different "starting account number" for teams at each site (for example, specify "Start Account Number at: 101" for teams at Site 1; specify "Start Account Number at: 201"

for teams at Site 2, etc.[68]). This will insure that the team IDs inserted into the XML Event Feed will be unique for each team across all contest sites, as required by some external tools.

[68] This example assumes you have at most 100 teams at any given site.

Appendix M – PC² Web Services

PC² contains an embedded web server designed to provide a variety of "RESTful" web services[69]. The embedded web server is disabled by default; it must be explicitly started by logging in using an Event Feed Account (see the Appendix on the **PC² Event Feed**). Logging into an Event Feed account displays the Event Feed Server interface as shown in the Event Feed Appendix; clicking the **Web Services** tab produces the following web server control display:



As shown above, the user can specify the port on which the web server listens (the default is 50443), as well as enable a variety of web services (REST resource endpoints). Clicking the Start button then starts the web server listening on the specified port and responding to requests for the enabled web services.

The embedded web server requires HTTP**S** connections and uses "BasicAuth" authentication. This means that connecting to the web server requires providing user/password credentials as defined by the BasicAuth specification.[70] The web server obtains credential information (for comparing against the user/password provided by connecting clients) by reading a file named **realm.properties**, which must exist in the folder from which the Event Feed login is started. (A sample **realm.properties** file is included in the PC² distribution; note however that all credential information in the sample file is commented-out.)

---

[69] See https://en.wikipedia.org/wiki/Representational_state_transfer for information on REST.

[70] See https://en.wikipedia.org/wiki/Basic_access_authentication. Note in particular that this means that the server responds to unauthenticated requests with a response whose header contains a *HTTP 401 Unauthorized* status and a *WWW-Authenticate* field, and that the client is expected to provide authentication in the form of an *Authentication* field constructed as described in the above-referenced specification.

Each non-comment line in the **realm.properties** file specifies three types of information: a *username*, the *password* associated with the specified user name, and a set of one or more *roles* associated with the user name. The general form of a **realm.properties** entry is

```
username: password,role1[,role2…]
```

The web server compares the user name and password provided by the connecting client with the set of username/password entries in the **realm.properties** file; if a match is found then the client connection is accepted and the client is assigned the role(s) specified on the matching line. The web server currently recognizes two "roles": *public* and *admin*. The role(s) associated with a client determine what services the user may access (see the table below).

A simple mechanism for verifying proper PC^2 web server operation is to start the web server and then point a browser to *https://<ip>:<port>/service* (where <*ip*> and <*port*> are the IP address and port where the web server was started and */service* is one of the REST endpoints listed in the table below) – for example, `https://198.1.100.0:50443/scoreboard`. This should return to the browser a request for credentials (username and password); entering valid credentials as specified in the **realm.properties** file should return the output corresponding to the specified service.

If the Event Feed client is started in "nogui" mode, the component's Graphical User Interface (shown above) is never displayed and therefore there is no way to interactively select the Web Services tab and start the web server. However, web services can be enabled in nogui mode by creating a file named "**pc2ws.properties**" in the Event Feed client's startup directory containing properties identifying what web services should be started. The **pc2ws.properties** file (example found in **samps/pc2wd.properties**) can contain the following entries to start the web server and enable the corresponding web services:

```
# port for web service
port=50443

#enable CLICS Contest API (defaults to enabled unless "no" is specified)
enableCLICSContestAPI=yes

# enable starttime web service
enableStartTime=yes

# enable submission_files web service
enableFetchRun=yes
```

See the PC^2 Wiki "Event Feed Module" entry (https://pc2.ecs.csus.edu/wiki/Event_Feed_module) for additional details.

The currently available REST services are defined in the following table (note that the first three comprise the PC^2 implementation of the CLI CCS JSON Scoreboard specification[71]). Additional REST services are planned for a future version of PC^2.

REST Endpoint	Roles Allowed Access	GET Response	PUT Response
/scoreboard	admin, public	The current contest scoreboard in JSON format	405 Method not Allowed
/teams	admin, public	A list of the teams in the contest in JSON format	405 Method not Allowed
/problems	admin, public	A list of the problems in the contest in JSON format	405 Method not Allowed
/languages	admin, public	A list of the languages in the contest in JSON format	405 Method not Allowed
/starttime	admin	A JSON String giving the scheduled contest start time as a Unix Epoch value, or the string "undefined" if no start time is currently scheduled	Resets the current contest scheduled start time according to the received (input) string, which is expected to be in JSON format as described in the CLICS Wiki "StartTime" interface specification

The following points should be noted regarding the PC^2 web server:

- The web server is designed to support requests for contest data services based on various CLI Specifications[72]. It is *not* intended to support web access to PC^2 by teams (see the separate Appendix on Team Clients for information on web access for teams).

- Web servers are inherently "resource intensive" (memory, CPU, etc.). Care should be taken regarding the machine on which the web server is run (it runs on the same machine as the PC^2 component which is used to start it). If the machine is going to be overloaded as a result of adding the web server, consideration should be given to running the web server on a separate machine. (This can be easily accomplished by starting the Event Feed client on the desired machine, using it solely to start and manage the web server.)

[71] See https://clics.ecs.baylor.edu/index.php/JSON_Scoreboard_2016

[72] See https://clics.ecs.baylor.edu/index.php

Appendix N – PC² Team Clients

PC² supports two different types of Team Client. The first is the traditional *Application Team Client* which has been standard since the first version of the system. With the Application Team Client, PC² must be installed on each team machine and each team machine must be provided with a **`pc2v9.ini`** file pointing to the PC² server, as described earlier in this manual.

Since PC² Version 9.7 the system supports an alternative Team Client known as the *Web Team Interface (WTI)*. The Web Team Interface was developed by students at Eastern Washington University, under the direction of Professor Tom Capaul.[73] The WTI is a *web-based* interface to the PC² server; it provides an *embedded web server* which listens for team *browser* connections and provides communication between the team browser session and the PC² server.[74] Teams use the browser for submitting runs, checking results, submitting and examining clarification results, viewing the scoreboard, etc. Since every modern machine installation includes a web browser, using the WTI means that *no additional software needs to be installed on the team machines to allow teams to connect to the PC² server.*

The WTI provides all the services provided by the PC² Application Team Client, with one exception: currently it does not support the "Test Run" facility.

By default the WTI server listens for connections at port **`8080`** on the machine on which is it running, and forwards information to the PC² server at the IP address and port specified in the **`[client]`** section of the WTI's **`pc2v9.ini`** file (the default values for the WTI's connection to the PC² server are **`localhost:50002`**). Both the IP address and port on which the WTI server listens for team (browser) connections and the address/port of the PC² server to which it connects can be adjusted, simply by editing the WTI's **`pc2v9.ini`** file. The WTI server does not need to be running on the same machine as the one on which the PC² server is running.

The WTI distribution includes a "User's Guide" for teams – a handout explaining what the WTI looks like and how to use it, analogous to the "PC² Team Guide" for the Application Team Client. The WTI User's Guide can be found in the "**`doc`**" folder of the WTI distribution (see below).

IMPORTANT NOTE: In order to support using the WTI, the PC² configuration must include a "scoreboard account" designated for use by the WTI. This account is used by the WTI to access current contest standings for the team display. Such an account must be created by the Contest Administrator when setting up contest accounts using the "Generate Accounts" function in the PC² Admin. By default the WTI expects to use account "**`scoreboard2`**" with password "**`scoreboard2`**", but this is configurable and any scoreboard account can be used. However, *the designated account SHOULD NOT BE USED FOR ANY FUNCTION OTHER THAN SUPPORTING THE WTI CLIENT*. If you plan to also run a separate "PC² Scoreboard", use a *different account* for that function. Note also that in any case, we *strongly* recommend you *change*

[73] Previous versions of PC² supported an older "web interface" client, called *EWTeam*, which was also developed by EWU students under Tom's direction. Documentation on the older EWTeam client is available in previous versions of this Guide, which can be found through the PC² website.

[74] PC² users who are familiar with the old *EWTeam* client will recognize that this is different: whereas the EWTeam required embedding PHP code into an *existing webserver*, the newer WTI client has its own self-contained webserver.

the PC² password for the scoreboard account to insure that no unauthorized individuals can log in using the (well-documented) "default password".

Prerequisites for using the WTI

- Team machines must have the ability to make an HTTP browser connection to the machine/port where the WTI server is running (see the next section regarding configurations where the WTI server is running behind a firewall or other component using Network Address Translation (NAT)) .

- The WTI machine must have the ability to make TCP/IP connections to the machine/port on which the PC² server is running.

- The PC² Admin must be used to create a scoreboard account to be used (exclusively) by the WTI. By default the WTI expects the Contest Administrator to create account "scoreboard2" for its exclusive use (see below). Note that we *strongly* recommend *changing the scoreboard account password!*

Setting Up the WTI

1) In the PC² distribution, go to the "projects" folder, copy the file **`WebTeamInterface-xx.zip`** (or **`.tar.gz`**) to any convenient location on any machine, and unzip it.

2) If needed, edit the **`pc2v9.ini`** file *under the* ***`WebTeamInterface-xx`*** *folder (NOT the* **`pc2v9.ini`** *in the PC² server folder)* as follows:

   a. If the scoreboard account which is designated for use by the WTI is anything other than "scoreboard2" with the default password (also "scoreboard2"), update the **`wtiscoreboardaccount=xxx`** and the **`wtiscoreboardpassword=yyy`** entries to contain the correct scoreboard account/password values. (See IMPORTANT NOTE, above.)

   b. If the WTI server is running on a *different machine* from where the PC² server is running, then in the **`[client]`** section, change the default **`server=localhost:50002`** entry to contain the IP address of the PC² server machine (instead of "**`localhost`**"). If the PC² server was started on a different port than the default (**`50002`**), update the port value in the **`[client]`** section **`server=`** entry as well.

   c. If the WTI server should listen for team browser connections on some port *other than the default (**`8080`**),* update the **`wtiport=`** entry in the **`[server]`** section to contain the desired port number.

   d. If the WTI server is running on a machine with a private IP (for example, behind a NAT firewall), add an entry to the WTI **`pc2v9.ini`** file of the form

wtiOverridePublicIP=w.x.y.z, where **w.x.y.z** is the public-facing IP address of the WTI server (that is, the IP address to which team browser sessions will initially connect). This is necessary to insure that connections *following* the initial contact by the browser will be directed by the WTI client code running in the browser to the correct (publicly-visible) IP address of the WTI server.

Starting the WTI

1) Go into the **WebTeamInterface-xx** folder.

2) Execute the command **./bin/pc2wti**.

This starts the WTI webserver running, listening for team browser connections on the specified (or default) port. At this point teams should be able to open a browser window pointing to the WTI. The URL which teams should use will be **http://<IP>:<port>**, where **<IP>** is the public-facing IP address of the machine on which the WTI server is running and **<port>** is the port specified in the WTI's **pc2v9.ini** file (default **8080**).

Appendix O – Input Validators

1. Overview

PC^2 allows the Contest Administrator to configure each contest problem so that it has associated with it one or more *input validators*.[75] An input validator is a program which is given, as input, a judge's data file (that is, a data file which a team's program is expected to read and process). The input validator program contains logic to make a determination, according to some set of rules, regarding the correctness of the judge's data.

PC^2 supports two types of input validators. First, PC^2 contains an embedded copy of the VIVA Input Validator.[76] VIVA supports a "pattern language" that allows a user to describe the required syntax of a judge's data file; PC^2 can be used to invoke VIVA with a specified pattern and display the results when the pattern is applied by VIVA to each contest problem judge's data file.

PC^2 also supports *custom* input validators. A custom input validator is an external program, written by the Contest Administrator (or staff), which accepts as input a judge's data file and returns an indication of whether the data file is "valid". Custom input validators can be written in any language whose compiler (or interpreter) can be invoked from a command line. PC^2 arranges that when a custom input validator is run, each judge's data file is passed to the custom input validator; PC^2 then displays the results.

Each contest problem can be configured so that it does (or does not) have a VIVA pattern, and can also be independently configured so that it does (or does not) have a custom input validator. Note however that running an input validator is a manual process (input validators are never run automatically; see below), so if a problem is configured with both a VIVA pattern and a custom input validator the user must manually choose which one is to be executed. (Both may be executed, but each one must first be 'selected'.)

Note also that while it is a bit of work to write an input validator (or even to write a VIVA pattern) to verify that all judge's test data files are valid, it can save a contest from disaster: if a last-minute change to a judge's data file inadvertently introduces an illegality in the data, running an input validator will catch this – otherwise, it might not be caught until well into the contest (or not at all!)

As with output validators (and virtually every other configuration item in PC^2), input validators can be configured either interactively or via YAML configuration files (see the chapter on *Configuring the Contest via Configuration Files* for details on YAML configuration files). The following sections describe interactive input validator configuration, and input validator configuration using a YAML file.

---

[75] As previously noted, the CLICS CCS specification defines two types of validators: *input validators*, which examine the judge's input data to insure the data complies with the specifications given in the contest problem statement, and *output validators*, which examine the output of a team's program for correctness. PC^2 supports both types of validators; this appendix is about *input* validators.

[76] VIVA is "Vanb's Input Verification Assistant". Permission to embed VIVA within PC^2 was kindly granted by David "vanb" Van Brackle, the author of VIVA.

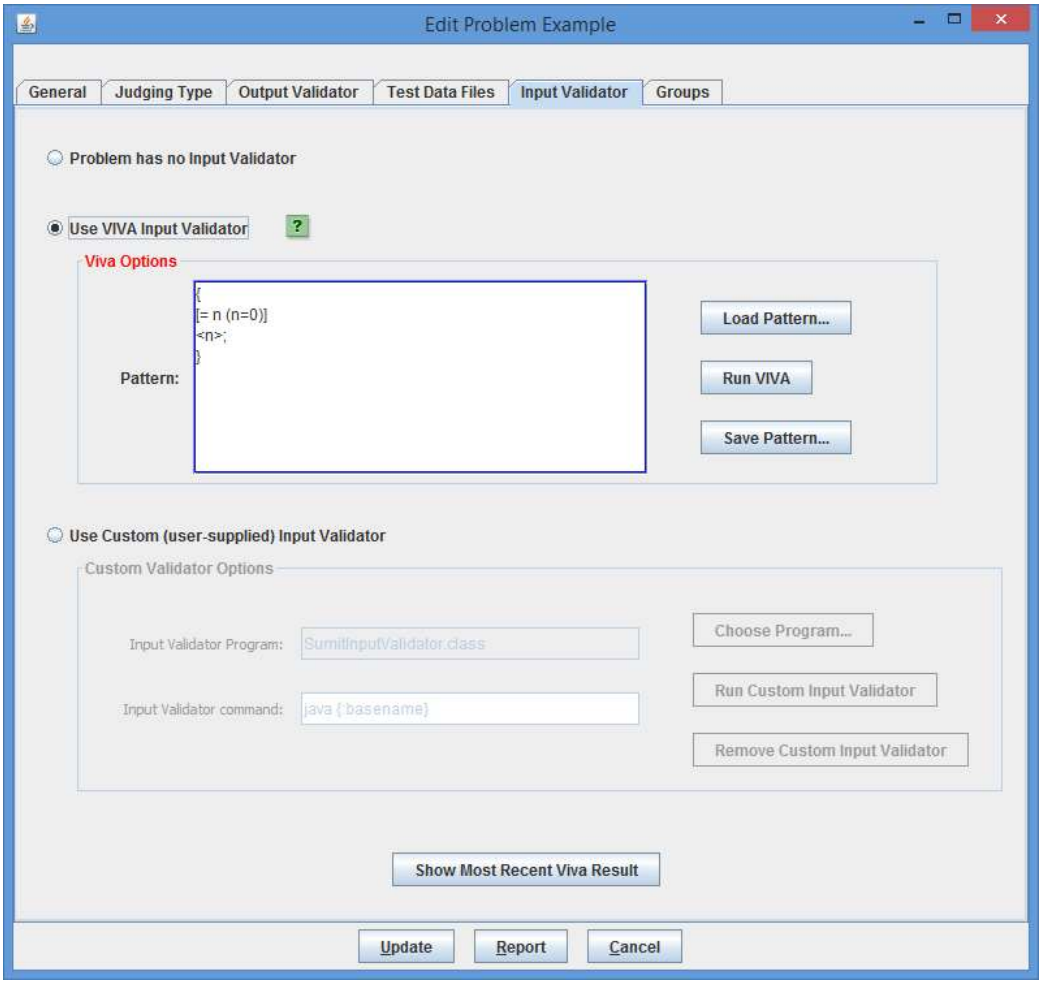
2. Interactive Input Validator Configuration

Input validators can be configured using the **Input Validator** tab on the Contest Administrator's **Edit Problem** or **Add problem** screens (the **Edit Problem** or **Add Problem** screens are accessed by clicking on the **Problems** tab of the Admin's **Configure Contest** screen, then clicking either the **Edit** or **Add** button). The same screen is used to configure (and invoke) both VIVA and custom input validators.

When any change is made on the **Input Validator** configuration tab, pressing the **Update** button saves the input validation configuration (along with any run results) as part of the problem configuration. Note that input validators are not run automatically in PC^2; the Contest Administrator must use the **Run VIVA** and/or **Run Custom Input Validator** buttons to test the judge's input data files with the corresponding input validator.

2.1. VIVA Input Validator Configuration

The following shows the **Input Validator** tab with the **Use VIVA Input Validator** option selected:

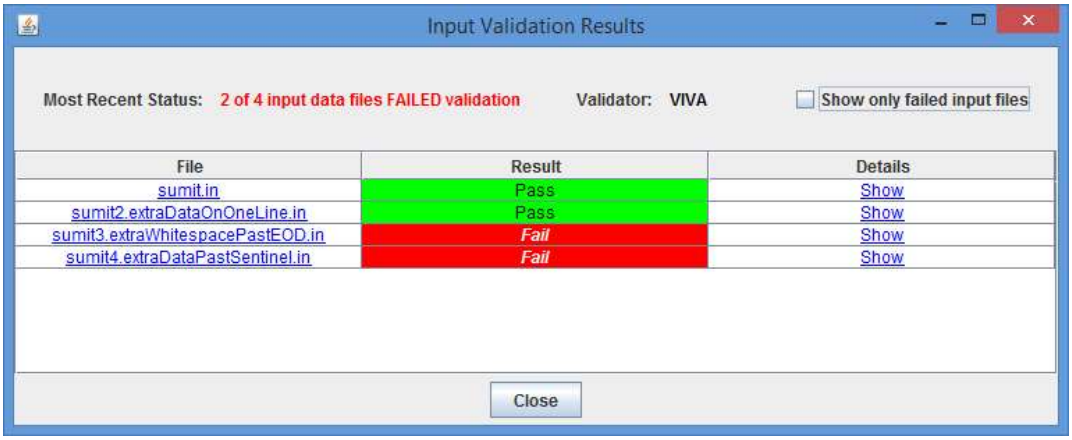


The example problem shown on the above screen has been configured with two input validators: VIVA as well as a custom input validator. Because the VIVA input validator is currently selected, this automatically enables the VIVA **Load Pattern**, **Run VIVA**, and **Save Pattern** buttons. The **Load Pattern** button has been used to load a VIVA pattern from a file (patterns can also be typed directly into the **Pattern** text area).

VIVA patterns are by convention stored in a file whose name ends with the suffix "**.viva**" and are placed in a folder named "**input_validators**" beneath the problem-specific folder (that is, beneath the folder named as the problem's "short-name") which in turn is stored beneath the contest "**config**" folder.

The VIVA pattern displayed above says that each judge's input data file must consist of a series of lines, each containing a single integer, and that the file data is terminated by a line containing a sentinel value of zero.[77]

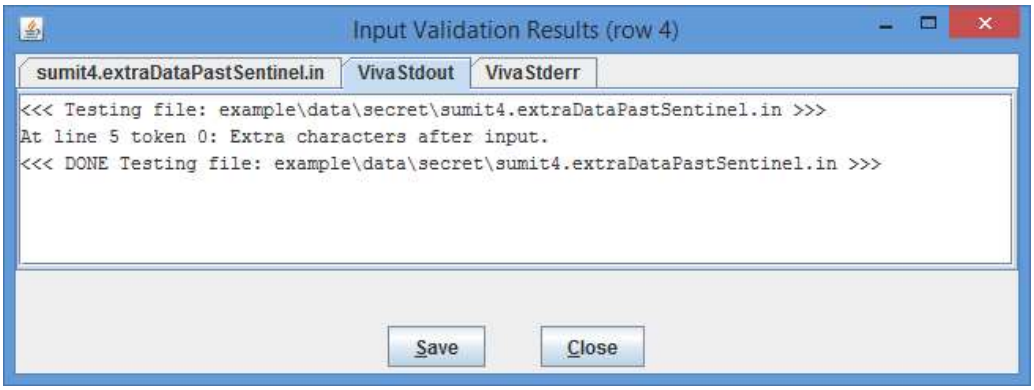
Pressing the **Run VIVA** button will cause PC^2 to send each judge's data file that has been configured into the contest problem (as shown on the **Test Data Files** tab) to VIVA, along with the current VIVA pattern, and then to display an "Input Validation Results" screen similar to the following:



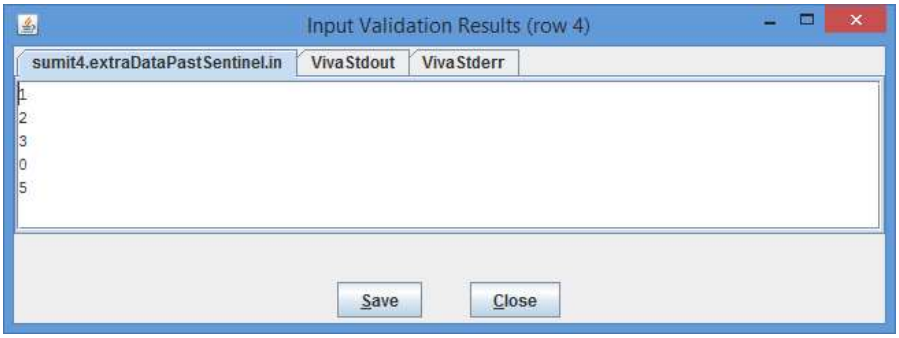
Note that in this example, four judge's data files have been configured on the **Test Data Files** screen; two of them passed VIVA validation while two failed to match the VIVA pattern.

[77] Refer to the *VIVA User's Guide*, which can be found in the "doc" folder under the PC^2 installation, for a complete description of VIVA patterns and how they work.

Clicking on any row in the Input Validation Results table will show details of the validation for the file listed in that row. For example, clicking on the fourth row produces the following dialog:



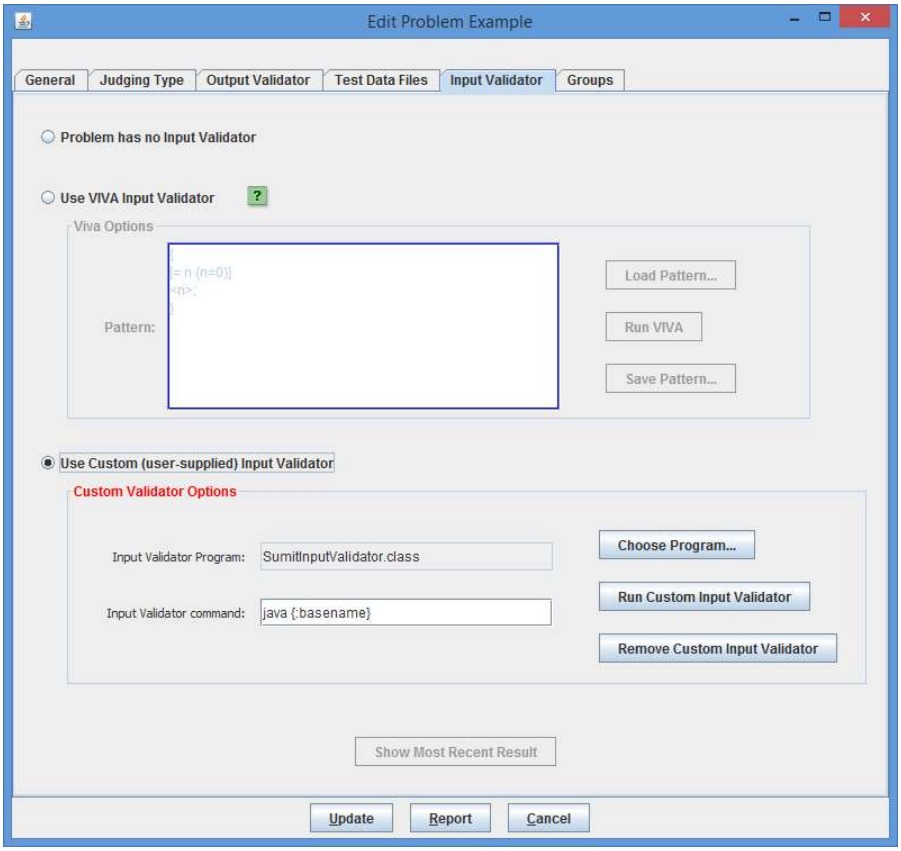
Note on the above screen that VIVA reports that there is extra data in the file after the expected input is completely processed. Clicking on the leftmost tab to display the file contents produces the following display:



This shows that the data file indeed has data following the sentinel value of zero (recall that the specified VIVA pattern says that the data file terminates with a line containing a zero).

2.2. Custom Input Validator Configuration

Selecting the **Use Custom (user-supplied) Input Validator** button on the **Input Validator** tab allows configuring a custom input validator:



Custom input validators require two configuration items: the name of the *input validator program* which is to be run, and the *input validator command* which is used to invoke the program. In the above example, the **Choose Program…** button has been used to select a Java program (**.class** file) named **SumitInputValidator.class**, and the input validator command has been set to invoke the Java JVM, passing to it the basename of the program (that is, the program name omitting the "**.class**" extension).

(Note: custom input validator programs, like VIVA pattern files, are by convention stored in a folder named "**input_validators**" beneath the problem-specific configuration folder under the "**config**" folder.)

Variable substitutions in the input validator command are supported as with other PC^2 command configurations, and PC^2 will automatically add an appropriate input validator command when it recognizes certain input validator programs (for example, the above input validator command "**java {:basename}**" was automatically added when PC^2 recognized that the input validator program was a "**.class**" file).

Pressing the **Run Custom Input Validator** button runs the custom input validator against each of the test data files configured in the problem and displays an Input Validation Results table analogous to the one shown above for VIVA.

A custom input validator can be written in any language, as long as the input validator command used to invoke it knows how to execute the input validator program (in other words, as long as the same command works correctly when typed at a command prompt). The input validator program should terminate with an Exit Code of 42 to indicate success (i.e., to indicate that the input data file passes validation). Any Exit Code other than 42 is interpreted by PC^2 to mean that the input data file failed input validation. (This follows the ICPC Problem Format specification, given at https://icpc.io/problem-package-format/spec/problem_package_format#input-validators.)

3. Input Validator Configuration via YAML Files

As with nearly all PC^2 configuration items, input validator configuration can be accomplished via YAML files. To configure input validator(s) via YAML, the **contest.yaml** file must contain a **problemset** YAML key with entries giving the name of each contest problem, and there must be a folder corresponding to each problem name beneath the **config** folder. Each problem folder beneath the **config** folder must contain a file named **problem.yaml**, and the **problem.yaml** file must contain a YAML section defined with the keyword **input_validator.**

The **input_validator** section of the **problem.yaml** file may contain any (or all) of the following keys, each of which would be followed by an equal sign and the value associated with that key:

YAML Key	Corresponding Value	Notes
defaultInputValidator	NONE, VIVA, or CUSTOM	Sets the default (currently-selected) Input Validator type
customInputValidatorProg	The name of the custom input validator program	Sets the custom input validator program name
customInputValidatorCmd	A string giving the command used to invoke a custom input validator	Sets the command to be executed when the Admin's "Run Custom Input Validator" GUI option is invoked.
vivaPattern	A string containing a VIVA pattern	Sets the VIVA pattern to be applied when VIVA is run
vivaPatternFile	The name of a file containing a VIVA pattern	If both vivaPattern and vivaPatternFile are specified, the vivaPattern is used

As an example, the following files might be used to configure a contest with a problem named "sumit":

Contest.yaml file stored in the **config** folder:

```
# Contest name:
name: Sumit Example
short-name: SumitEX
duration: 5:00:00
scoreboard-freeze-length: 1:00:00
languages:
  - name: Java
    active: true
  - name: GNU C++
    active: true
problemset:
  - letter: A
    short-name: sumit
    color: yellow
    rgb: "#FFFF00"
```

Problem.yaml file stored in folder "**sumit**" beneath the "**config**" folder:

```
name: Sumit
timeout: 10
input_validator:
   defaultInputValidator: custom
   vivaPattern: '{x;}'
   customInputValidatorProg: 'SumitInputValidator.class'
   customInputValidatorCmd: 'java {:basename}'
```

The above problem.yaml file specifies that the problem named "Sumit" (which is stored in folder "**sumit**") has a 10-second run time limit and has a default input validator type of "custom", a VIVA pattern of "**{x;}**", and a custom input validator program named "SumitInputValidator.class" which is invoked using the command "java SumitInputValidator". (Note that the above is not a complete problem.yaml file; it is only intended to show how to configure input validators. Typically the problem.yaml file would contain other entries as well, such as the *output* validator configuration, information on where team programs should read their input (for example, from stdin or from a file), etc.)

Note that the "**sumit**" folder beneath the "**config**" folder should also contain folders "**data**", holding the "sample" and "secret" data files in correspondingly-named subfolders; "**input_validators**", holding the "**SumitInputValidator.class**" file; and "**problem_statement**", holding a LaTex copy of the problem statement containing at least a **\problemtitle{Sumit}** statement giving the problem title. See the CLICS Contest Data Package specification for more information on the structure of a Contest Data Package (CDP).

Appendix P – reject.ini

PC^2 allows the Contest Administrator to initially configure judgements using a **`reject.ini`** file. This file must be present on the server machine in the installation directory (the directory that has the pc2 bin/ directory) or (since Version 9.7) in the folder containing the *Contest Data Package (CDP)* which is used to configure the contest. These judgements are loaded once on initial server startup. These judgement names and acronyms can be changed at any time using the Admin from the Judgements tab under the Configure Contest tab.

Each judgement text from the **`reject.ini`** will be loaded and prepended with a "No - " phrase. There is one exception: a line that has an AC judgement acronym will not have a "No – " phrase prepended.

File Format and Contents

Blank lines and lines starting with # are ignored.

Each line contains a judgement and optional judgement acronym delimited by a vertical bar (|), for example:

```
Time Limit Exceeded|TLE
```

This would produce a judgement with the text "No -Time Limit Exceeded"

Example 1 – reject ini judgements and acronyms

```
# reject.ini with acronyms
Compilation Error|CE
Run-time Error|RTE
Time Limit Exceeded|TLE
Wrong Answer|WA
Excessive Output|EE
Output Format Error|EFE
Various Differences|VD
Other - Contact Staff|CS
Yes, yes, yes|AC
```

When loaded that will produce the following judgements:

Note that the AC acronym line from the reject.ini did not have the "No – " prepended.

```
Yes, yes, yes
No - Compilation Error
No - Run-time Error
No - Time Limit Exceeded
No - Wrong Answer
No - Excessive Output
No - Output Format Error
No - Various Differences
```

```
No - Other - Contact Staff
```

Note that if the judgement acronym is AC then that will replace the Yes judgement text.

Example 2 – reject.ini with judgements only

```
# reject.in with judgements but no acronyms
Compilation Error
Run-time Error
Time-limit Exceeded
Wrong Answer
Excessive Output
Output Format Error
Other - Contact Staff
```

When loaded that will produce the following judgements:

```
Yes
No - Compilation Error
No - Run-time Error
No - Time Limit Exceeded
No - Wrong Answer
No - Excessive Output
No - Output Format Error
No - Other - Contact Staff
```

For additional information refer to the Wiki Article https://pc2.ecs.csus.edu/wiki/Reject.ini

Appendix Q – GUI Customization

Starting with Version 9.7, PC^2 allows *customizing* the appearance of the PC^2 login screen by adding your own University or Club logo and your own contest-specific banner to it.

Customization is accomplished by creating a new folder named "`images`" in the PC^2 installation folder (that is, in the folder containing the PC^2 `bin`, `data`, `doc`, `lib`, and `samps` folders) and then adding appropriate image files to the "`images`" folder. Be sure that both the `images` folder and the image files within it are readable.

When PC^2 starts, it looks for files named "`logo.png`" (or "`logo.jpg`") and "`banner.png`" (or "`banner.jpg`") in the "`images`" folder. If a "`logo.png`" or "`logo.jpg`" file is found, PC^2 replaces the default University logo on the login screen with the image found in that file. If a "`banner.png`" or "`banner.jpg`" file is found, PC^2 replaces the default ICPC banner at the bottom of the login screen with the image found in that file. (If both a `.png` and a `.jpg` file are present, the `.png` file is used.)

Care should be taken to provide logo and banner files with proper aspect ratios. Logo files should be (approximately) square, while banner files are typically several (4-10) times as wide as they are high. Any provided logo file will be automatically scaled to be square and to fit in the available login screen logo area (approximately 130x130 pixels); likewise, any provided banner file will be automatically scaled to fit in the login screen bottom banner area (approximately 750 pixels wide by 70 pixels high). User-provided images should be designed with these sizes in mind.

Note that the `images` folder/files must be present on each PC^2 client machine. If you want all your PC^2 clients to have the same customized appearance, we suggest you unzip your initial PC^2 installation, add the images to this master configuration, then re-zip the system and copy the new zip to all your clients. (Alternatively, just copy the `images` folder/files to each client machine.)

The following shows a login screen which has been customized with (somewhat artificial) user-provided logo and banner files:

Appendix R – Shadow Mode

1. Overview

Starting with Version 9.7, PC^2 supports the ability to run in so-called *shadow mode.*[78] Shadow mode is a mechanism whereby the PC^2 system, rather than directly managing a contest, is instead used to *verify the results of a contest being managed by a different contest control system.*

Running a contest control system in shadow mode (or just "running a Shadow", as it's usually referred to) is regularly used in competitions such as the ICPC World Finals, as well as at many ICPC Championships and other contests. In such competitions, there are *two independent* contest control systems used during the contest – one called the *Primary CCS* and a second called the *Shadow CCS*. Teams communicate (only) with the primary CCS, sending submissions to it and getting back judgements and standings information from it.

Submissions are also sent (without teams really being aware of it) from the primary to the shadow CCS. This is accomplished by having the shadow login to the primary CCS through a defined interface. In particular, PC^2 expects to communicate with the primary CCS (the CCS which it is shadowing) through a standard "Contest API" defined by the *Competitive Learning Initiative Contest Systems* group (CLICS). The specification for the CLICS Contest API can be found at https://clics.ecs.baylor.edu/index.php?title=Main_Page.

Once logged in to the primary CCS, the shadow PC^2 system uses the CLICS API to fetch team submissions from the primary and then executes those submissions *just as if they had been sent directly to the shadow by the team*. The shadow then uses the execution results to compute contest standings. Contest results computed by the shadow are then compared with those computed by the primary; in this way it is possible to have *independent verification* that the contest results are correct.

Note that while it is theoretically possible to have PC^2 shadow *itself* (and this is sometimes done for testing purposes), the real reason for using shadow mode is to have PC^2 provide independent verification that the results computed by some *other* Contest Control System implementation agree with the results computed by PC^2.

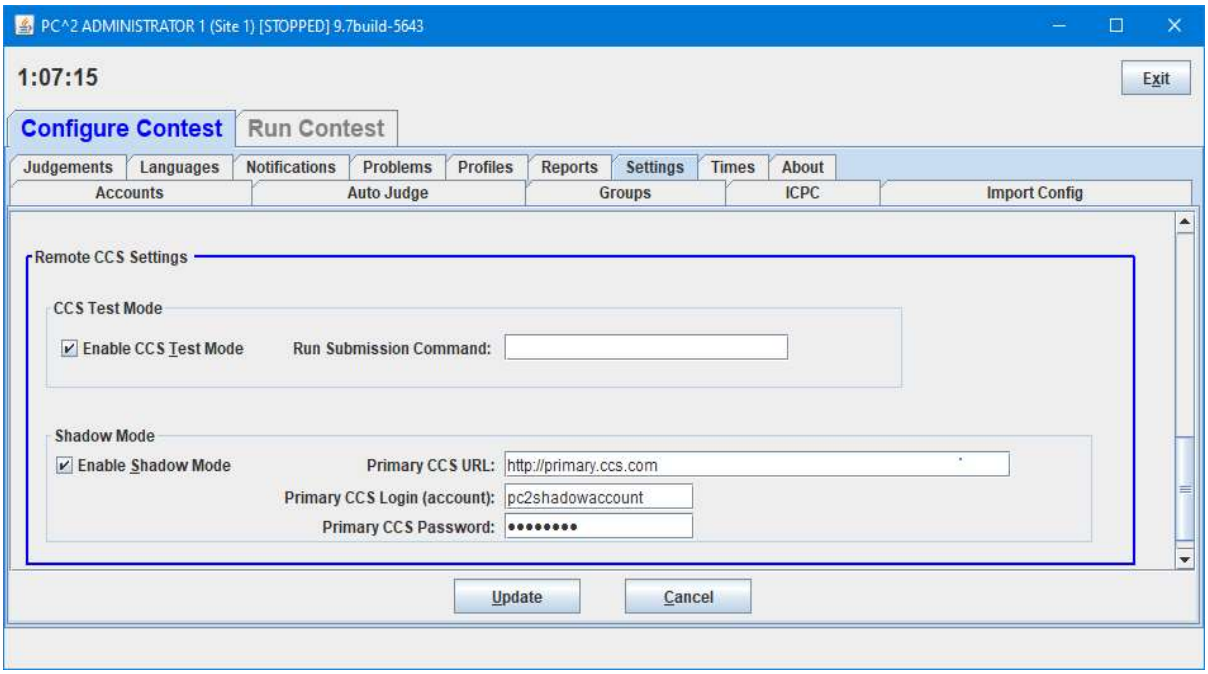
2. Shadow Setup

PC^2 can be configured to run in shadow mode using either of two mechanisms: *interactively,* or via *YAML configuration files.* The following sections describe how to set up a PC^2 system using these approaches.

2.1. Interactive Setup

Interactive shadow mode configuration is controlled through the PC^2 Contest Administrator's **Configure Contest > Settings** screen, as shown below:

[78] Shadow mode support was actually introduced in PC^2 Version 9.5, but that support was limited and relied on a deprecated interface; the current implementation is much more general and robust.



(Note that the **Settings** tab has been selected and *scrolled down*; the shadow mode configuration settings are contained within the **Remote CCS Settings** section at the bottom of the screen.)

To configure shadow mode, enter the URL which the PC^2 shadow will use to connect to the primary CCS, along with the account name and password which PC^2 will use to login to the primary CCS. Note that, as mentioned above, PC^2 expects the primary CCS to support the CLICS Contest API.

To enable shadow mode, check the **Enable Shadow Mode** checkbox. *In addition, check the* **Enable CCS Test Mode** box (this box is a legacy configuration item which must be enabled in order for shadow mode to work). Finally, click the **Update** button to load the shadow configuration into PC^2.

2.2. YAML Configuration

As with almost all other configuration items in PC^2, shadow mode can be fully configured via a YAML file (see the chapter on **Configuring the Contest via Configuration Files**). In particular, the following YAML keys can be included in a configuration file to configure PC^2 for shadow mode:

```
shadow-mode: true
ccs-test-mode: true
ccs-url: <URL for primary CCS>
ccs-login: <login account on primary CCS>
ccs-password: <primary CCS account pw>
ccs-last-event-id: <event id from which shadowing should start>  (see below)
```

2.3. Shadow Contest Configuration

In addition to configuring the PC^2 system (either inactively or via YAML files) for shadow mode, PC^2 must have a *contest configuration* which matches the configuration in the primary CCS. For example, PC^2 must be told what *languages* and *problems* are being used in the contest, what *teams (accounts)* are competing, etc. All of this configuration needs to be set up prior to starting shadowing operations.

The standard PC^2 contest configuration mechanisms are used to accomplish shadow contest configuration. The Contest Administrator must obtain the contest configuration details from the group that configures the contest in the primary CCS. (If the primary CCS is using a CLICS "CDP" to configure their contest, that CDP can be loaded into PC^2 as well (see the Chapters on Contest Configuration)). In any case, at least the contest *languages, problems,* and *team accounts* must be configured in PC^2 to match the primary CCS in order for shadowing to work correctly.

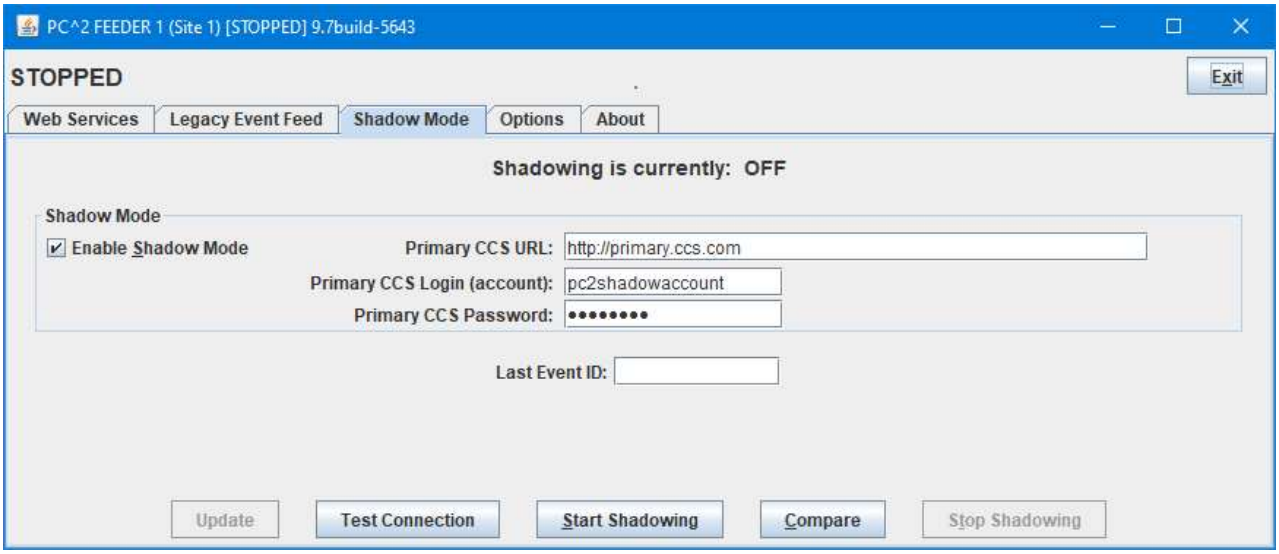
In addition to configuring the shadow contest to match the primary CCS, shadowing is really only practical if PC^2 is configured for "Auto-Judging" the contest problems. (Otherwise, variations in human judging could cause the shadow results to be different from those computed by the primary CCS.) This means that the following additional steps should be included when setting up a PC^2 shadow:

1) Configure all problems for "Computer Judging".
2) Configure one or more "Auto-Judge" accounts so that all contest problems have at least one "Auto-Judge" assigned to them.
3) Start one or more "Auto-Judges".

3. Starting Shadowing

The above steps *configure* a PC^2 system for shadow operations, but they do not actually start the system performing shadowing. To start shadowing it is necessary to run a PC^2 **Event Feeder** module. (The **Event Feeder** is a composite module which supports a variety of external communication functions for a PC^2 system; see the Appendices on the **PC2 XML (Legacy) Event Feed** and on **PC2 Web Services** for additional features of the **Event Feeder** module.)

In order to run a PC^2 **Event Feeder** there must first exist a **Feeder** account with which to login; at least one such account must be created using the PC^2 Admin **Accounts > Generate** function. Once a **Feeder** account has been created, an **Event Feeder** module can be started by executing the command `./bin/pc2ef` and logging in using the **Feeder** account. This will display the following screen (note that the **Shadow Mode** tab has been selected):



This screen displays the current shadow mode values (as configured either interactively or via a YAML file). It also displays an additional field, **Last Event ID**. This field can be used to inform the primary CCS that the shadow wishes not to start from the very beginning of the contest, but rather is only interested in events which have occurred *since* the specified **Last Event ID**. This is useful when restarting a shadow which has already fetched and executed large amounts of data from the primary; it avoids having to repeat the fetch/execute cycle for submissions already processed by the shadow. The format of the **Last Event ID** must match the format of event IDs sent from the primary CCS.

One thing to note is that this screen does *not* provide for enabling the legacy **CCS Test Mode** state, which is required for shadow operations to proceed. This is one reason why it is necessary to either configure shadow mode interactively (and check the **CCS Test Mode** box), or to specify `ccs-test-mode: true` in a YAML file.

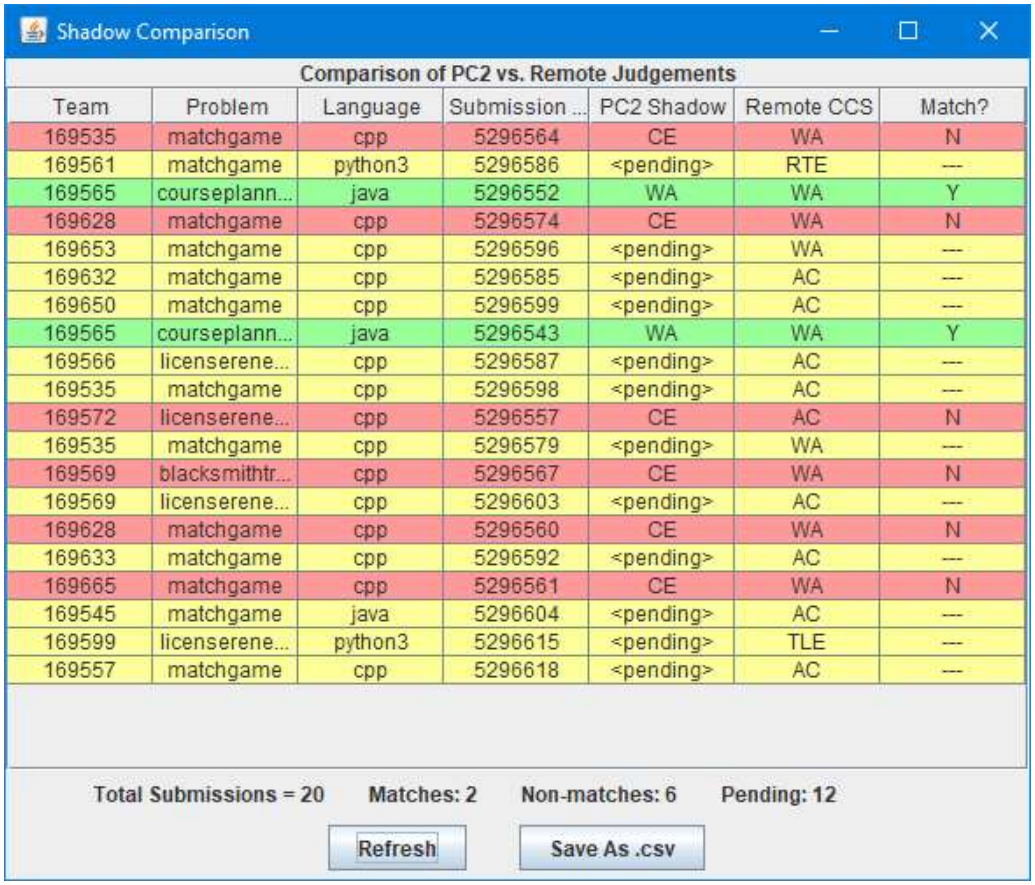
The **Test Connection** button can be used to verify that the PC^2 shadow can establish a connection to the primary using the specified URL; pressing **Test Connection** will return a failure message if no connection could be established. Note: **Test Connection** only verifies that *there is a valid network path to the specified URL*. It does *not,* for example, actually login to the remote (primary) system, validate credentials, or similar operations.

Pressing the **Start Shadowing** button will cause PC^2 to login to the primary CCS and begin fetching and (presuming an Auto-Judge has been started) executing submissions.

4. Comparing Shadow Results

Starting a PC^2 scoreboard (as described elsewhere in this manual) will allow viewing the PC^2-computed results for comparison with the primary CCS. However, this is really only practical at the *end* of a contest (otherwise, it's hard to grab the primary and shadow scoreboards at exactly the same moment/state).

A much more useful method of comparing what the shadow is doing with respect to the primary is to press the **Compare** button after pressing **Start Shadowing**. Assuming (1) a successful connection to the primary CCS, (2) that submissions exist in the primary CCS, and (3) that PC^2 Auto-Judging has been enabled, the **Compare** button will produce a screen similar to the following:



This screen shows that currently the PC^2 shadow has fetched a total of 20 submissions from the primary CCS; 2 of them (green) "match" (that is, PC^2 arrived at the same judgement as the primary), 6 (red) did not match (PC^2 arrived at a *different* judgement), and 12 submissions (yellow) are "pending" (still being judged by PC^2).

Note that this is an artificially-created example; it contains more "non-matches" than would normally be expected – but it shows for example how there might be a problem between the shadow and primary configurations (specifically, since all 6 "non-matches" are for submissions using C++ and the assigned judgement in all cases was "CE" (compiler error), there might be a configuration discrepancy with respect to the configuration for the C++ language).

The **Compare** screen can be sorted on any column by clicking in that column's header. The screen does not automatically update, but it can be refreshed (updated with the current status) by clicking the **Refresh** button. Clicking **Save As .csv** allows saving the current results in **.csv** format in a selectable file.