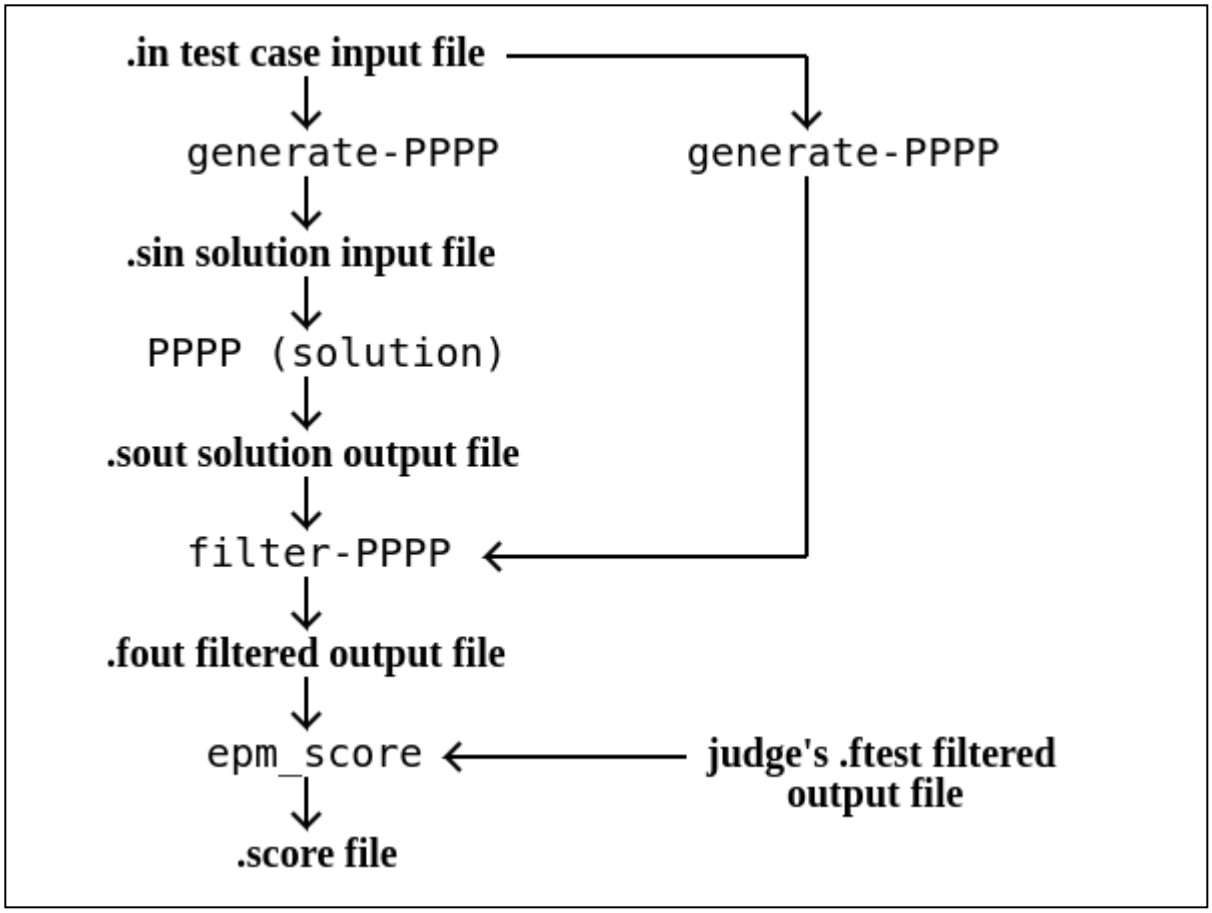
EPM Help Page

Introductory Note

EPM (the Educational Problem Manager) is intended to make it easier for one user to develop a programming problem and then make it available for other users to solve. Problem development is mostly a matter of creating test case input and output in a form suitable for automatic scoring, and then debugging the problem solution and specification against the test cases.

The following diagram specifies the data flow used to score a test case for a problem named PPPP. For simple problems, generate-PPPP simply copies its input to its output removing comments, and filter-PPPP simply copies its input to its output (epm_score ignores comments). In more complex cases generate-PPPP uses commands from the .in file to generate .sin solution input using a pseudo-random number generator, and filter-PPPP analyzes the .sin and .sout files to produce a summary used in scoring (e.g., it might read a graph from .sin and check that a path in .sout is in fact a path in the graph, outputting the path length if it is and an error message otherwise).



Login Page

To log into an EPM account, you open the URL of the EPM server you wish to access and load its **Login Page**. This page accepts a **login name** of the one of the forms:

User-E-Mail-Address	User Account
Team-ID:Member-E-Mail-Address	Team Member in Team Account
User-ID:Guest-E-Mail-Address	Guest in User Account

EPM has two kinds of accounts: user accounts and team accounts. These behave similarly in most circumstances. Both kinds of accounts are named by an **account ID**, which is either a **user ID** or a **team ID**. E-mail addresses map to user IDs, and a single user can have several e-mail addresses. Teams have members, which are user accounts. A user may have guests, which are other users who are allowed to see the user's account but not change it. Guests and teams are discussed separately.

An account is on an **EPM server**. EPM servers are named by the document root URL used to access them, and EPM servers with different names are independent of each other (even if they are running on the same computer).

Once you have logged in, an EPM server **session** for your login name is attached to your browser, and a ticket for future use with the login name is stored in your browser local memory. The ticket enables you to log in again using the same login name and browser without having to receive a confirmation number by e-mail.

Your session is allocated:

- A **main tab** for non-problem-specific actions. This is allocated on login and after login is loaded with the Project Page. You should not close this tab while you are logged in, but if you do you can recover it by re-using the EPM server URL you originally used to open this tab.
- A **problem tab** for each of your problems. These are created using the Project Page. There can be at most one problem tab for each of your problems.
- An **auxiliary window** for read-only views of files and information. Views of read-only information are placed by default in this window, but if you hold down an ALT key while launching a view that would normally go in the auxiliary window, a separate floating window will be created instead that is dedicated to the content of the particular view (e.g., this allows each file being viewed to have its own window).
- A **help window** for this Help Page and the User Guide page.
- A **documents index window** holding an index of available documents, and a **downloads index window** holding an index of downloadable example files.

EPM consists of a set of pages, such as the Login Page and the Project Page. Each of these is assigned to a tab or window. The Table of Contents gives the tab or window of each page in [] brackets after the entry for the page; for example, after the Login Page one sees [main], indicating that the Login Page runs in the `main' tab.

If you close the main tab by mistake, create a new tab and load it with the EPM server URL (that you used for logging in). Do NOT attempt to use an analogous method to create any other tabs or popup windows for your login session; use the buttons that create these instead.

DO NOT USE the browser Back or Forward buttons, as these will invalidate your tab. You may use the refresh keys, or the ↻ refresh button next to the ? help button. Many pages can be put into an editing mode in which the ↻ button disappears and the refresh keys are disabled.

To change the account ID of a session you must explicitly log out, and then you must log back in. You can log out using the `Logout' button on the User Page which should close any problem tabs and popup windows that you have. Or you can simply close and reopen your browser.

User E-Mail Address

You can have up to 3 **e-mail addresses** that map to your user ID, and hence to your account on the EPM server. Confirmation numbers will be sent to these e-mail addresses in order to create Browser Tickets.

If you are making a new account, enter an appropriate e-mail address. After entering the confirmation number sent to that address, you will be routed to the User Page to complete the process of establishing a new account.

Browser Ticket

A browser **ticket** is stored in the browser local memory and enables an account to log in by giving only the login name associated with ticket. There is a separate ticket for each EPM server, each browser, and each login name.

When you first use a login name with your browser, your browser has no ticket. Therefore the server e-mails you a ticket called a **confirmation number**, which you enter into the browser. The browser sends this to the server, which logs the browser in.

Every time the browser sends a ticket to the server, if the server accepts the ticket and logs the browser in, then the server sends the browser a replacement ticket. A ticket can be used only once. Should the server reject a ticket, it will give you another confirmation number.

When you first log using a given e-mail address, your first **auto-login period** for that e-mail address begins. During this period the server accepts tickets associated with the e-mail address from browsers that have them.

At the end of this period the server e-mails you a confirmation number when you next log in using the e-mail address, and starts a new auto-login period when it receives the confirmation number. The first auto-login period is 2 days, the second is 7 days, and all the rest are 30 days.

User Page

To be an EPM user you must supply the following information which appears on the User Page.

NOTICE: All this information is PUBLIC in that it is available to other users of the same EPM server, except that current policy is that the non-domain parts of e-mail addresses will be replaced by dots(...), e.g., jdoe@gurgle.com will become `...@gurgle.com'.

User ID:
A short name by which you will be known. Must consist of letters, digits, underline(_), and dash(-), begin a letter, end with a letter or digit, and have at least 4 characters. For privacy, you may wish to choose an ID that is NOT part of your e-mail address. NOTICE: Once you have created your account, you CANNOT CHANGE your user ID!

Full Name:
Your full name, so that you can be looked up on the web.

Organization:
An organization you are associated with, so you can be looked up in the context of this organization. If you are not closely associated with an organization, such as a university, you may use an organization such as the ACM or if nothing more limited is available, Facebook.

Location:
The location of your organization, e.g., town, state, and country, or if the organization is nebulous, like the ACM or Facebook, your own location.

E-Mail Addresses:
You must have one or more e-mail addresses associated with your account. You use these to log into the account. It is well to have two or more, in case you lose access to one of them.

Guests:
You may have guests which are other user accounts that can see but not change the contents of your account. A guest account is specified by giving the user ID of the account.

Guests

If you are a guest of another user, you may log in as a guest using a login name of the form:

Host-ID:E-Mail-Address

where Host-ID is the user ID of the user whose guest you are (your `host') and E-Mail-Address is one of the e-mail addresses associated with your own account.

Your guest login session is logged into the host account, but will only be able to see the contents of the host account, and will not be able to change these contents.

When you are logged in as a guest, you cannot edit your own profile or the profile of teams of which you are a manager, and you cannot create new teams.

Teams

A team account is like a user account except that:

1. A team account has **members** instead of e-mail addresses. Members are user accounts named by user IDs or by e-mail addresses that map to user IDs.
2. All the members of a team account can log into the account at one time. But only one of these login sessions can be read-write; the rest must be read-only.
3. When you log into a team account, you give a login name of the form:

   Team-ID:E-Mail-Address

   instead of just an e-mail address. The Team-ID in this login name is the account ID of the login session; the e-mail-address identifies you and specifies the user ID associated with the login session.
4. A team also has a **manager**, which is the account that can edit the profile of the team and add or delete team members.

Because a team account session is like a user account session in most ways, in most contexts EPM documentation uses the term `**user**' to mean either a user account or a team account.

When you are logged in as a team member there is a button in the upper right corner of all main tab pages, next to the help button, that you can use to switch your login session from read-only to read-write and back. This button is labeled `**RW**' to switch from read-only to read-write, or `**RO**' to switch from read-write to read-only.

At most one login session at a time for the team account can be read-write. A read-write session must be switched, by using the RO button, to read-only, so that all account sessions are read-only, before an session can be switched using the RW button to read-write. All the people logged into the same team account at one time should have open-mike audio communications, or a text-based equivalent, to make this work.

However there is an exception to this last paragraph to handle the case where a team member has forgetfully left their login session in read-write mode. The **User Page** RW button is special, in that it will allow you to force the switch from read-only to read-write even if another user is read-write, by forcing that other user into read-only mode.

Any user can make a team using the User Page. A team has a profile like a user profile, except that a team also has a **manager**, that is the user account that can edit the profile and add or delete team members. New team members can be identified by either their user ID or by one of their e-mail addresses. A manager can change a team's manager to another manager.

It is possible to identify a new team member using an e-mail address that is not yet associated with a user account, and the associated user ID will be determined automatically when the e-mail address is associate with a user account. This allows team members to be added to the team before they have established user accounts on the EPM server.

When you are logged in as a team member, you cannot edit your own profile or the profile of teams of which you are a manager, and you cannot create new teams, even if you are in read-write mode.

Project Page

A project is a collection of problems which may be solved by EPM users. Before solving a problem in a project, a user pulls the problem from the project. A user may also write their own new problem that does not belong to any project, and when done, the user may elect to push that problem to a project so other users can pull the problem from the project and try to solve it.

Two builtin projects are `**demos**', which contains demonstration problems, and `**public**', into which anyone may push problems they have developed.

The Project Page lets you:

- view descriptions of problem lists
- display or download descriptions problems in a problem list
- pull a problem you want to solve from a project
- create a new problem of your own that is not in a project
- create a problem tab for any of your own problems (whether they have been pulled or created by you)
- push a problem you have written into a project

The description of the current problem list is displayed if it exists. To display the description of a problem in the current problem list, click on on the problem name at the bottom of the page. To download the description, click on the problem name while holding the control key down.

To pull a problem from a project, first select the problem list of the project, which will have a name such as `public Problems' or `demos Problems'. Then click the **Pull** button, check the ovals of the problems you wish to pull, and click the **Submit** button. Details are at Pulling Problems From Project.

To create a tab for one of your own problems, first select a problem list that contains the desired problem, such as the `Your Problems' list that contains all your problems. Then select the problem from the `**Create Tab for**' problem list. `Create Tab for' only exists of the current problem list contains some problems that you previously created or pulled.

To create a new problem of your own, simply type its name into the `**Create New Problem**' box. The name must consist of letters, digits, underline(_), and dash(-), begin with a letter, and end with a letter or digit. If you plan to publish this problem in a project, you should choose a name distinct from other problem names already in the project.

You should understand that when you create a problem of your own, the master copies of the problem files remain on your own computer. These files are uploaded to the EPM server, but the uploaded copies might not be preserved.

Pushing problems is done in a manner similar to pulling them: see Pushing Problems to Project for details.

When you pull or push a problem, the problem ends up in both a directory private to you, as `Your' problem, and in a directory within a project. The project problem directory is known as the **parent** of your problem directory. Your problem directory will then contain symbolic links to files in its parent. However, solution program files are only in your problem directory, and NOT in its parent.

You can make your own problem lists, and publish them so others can use them. To do this click the `**Edit Lists**' button on the Projects Page. See List Page for details.

The list of problem lists on the Projects Page is known as the Favorites List. This Favorites List can be edited to include or exclude various problem lists by clicking the `**Edit Favorites**' button on the Projects Page. See Favorites Page for details.

You will likely not have access to all projects. You will be able to pull problems from `demos', but NOT push problems to it. You will be able to both pull problems from and push problems to `public'. Projects you have no access to will not be visible on the Projects Page. See Manage Page for details.

Pulling Problems From Project

If you want to solve a problem that someone else has written, you must pull the problem from a project. Pulling a problem creates a local directory for the problem, that is, your own private local directory for the problem, and then creates links in this directory to files in the project problem directory. Only some files are linked in this manner. You have to upload a executable solution to your local directory to do things like making score files.

Note that if you wrote and pushed the problem, so the project problem is yours and not someone else's, all the project problem files will be linked into your local directory, rather than just those you should have to try to solve the problem.

To pull problems, in the Project Page select a problems list and click `**Pull**'. Then check the ovals for the problems you want to pull. Lastly, click **Submit**. If all of the problems pull successfully, the pulled problem ovals will turn green, a `**Done!**' message will appear, and when you have finished looking at the colors of the problem ovals, you can click `**Finish**' or go to one of the other pages that share the `main' tab with the Project Page.

Checked problems are pulled in the order they appear in the list. If there is an error or warning pulling a problem, messages will be displayed and you will be given a choice on how to proceed. You can skip to the next problem or cancel all pulling. For warnings, you can ignore the warning and execute the pull actions.

If you want to see the actions that will be taken to pull a problem, check the `**Check Proposed Actions**' button before you `**Submit**'. You will be shown the list of actions needed to pull the problem and asked whether or not to execute them or skip to the next problem or cancel all pulling. The problem's oval will become red while you are deciding what to do about the actions.

At the end a problem's oval will be green if it was successfully pulled, yellow if the problem was skipped because errors were detected in compiling the problem's pull actions (no actions were actually performed), and white if the problem was skipped over.

Pushing Problems to Project

If you have written your own problem, you can make it available for others to solve by pushing the problem into a project, such as the public project. Then others can pull your problem from the `public' project and try to solve it.

To push a problem to a project you must have `push-new' privilege for the project. You may re-push a problem you have previously pushed in order to update it. To re-push you must have `re-push' privilege for the project problem. When you initially push a problem, and thereby create the project problem, you are given `owner' and `re-push' privileges for the project problem. See Manage Page for details.

When you push a problem, files in your local problem directory are moved to its parent project problem directory and links are made in your local problem directory to these moved files. One consequence of this is that all runs you do for the problem will be local, and you will not be able to submit the problem. However your local runs give results identical to submissions (they just are not recorded as submissions), as long as you have not replaced any local links by updated files. If you do replace local links by updated files, you may want to re-push after you have finished testing the updates.

If you delete a file link in your local problem directory whose target is a file the parent directory, executions are likely to still work as in most cases a file will be taken from either the local directory or the parent, with the local directory preferred for non-submissions, and the parent version required for non-solution-executables for submissions. If you delete a local problem directory link of a .run file, you will find that you can now submit the file. If you want to restore local links you have deleted, you may do so by pulling the problem.

To push problems, in the Project Page select a problems list and click `**Push**'. Then check the ovals for the problems you want to push and if some of these problems have not previously been pushed, use `**Select Project**' to select the project to push the problems into. Lastly, click `**Submit**'. For each problem you will be shown the list of actions needed to push the problem and you will be given a choice as to whether to execute these or skip the problem or cancel all pushing. While you are deciding, the oval of the problem being processed is red. If you do not want to see these actions and be given this choice, uncheck the `**Check Proposed Actions**' oval.

If there is an error or warning pushing a problem, messages will be displayed and you will be given a choice on how to proceed. You can skip to the next problem or cancel all pushing. For warnings, you can ignore the warning and execute the push actions.

When all of the problems to be pushed have been processed, a `**Done!**' message will appear, and when you have finished looking at the colors of the problem ovals, you can click `**Finish**' or go to one of the other pages that share the `main' tab with the Project Page.

At the end a problem's oval will be green if it was successfully pushed, yellow if the problem was skipped because errors were detected in compiling the problem's push actions (no actions were actually performed), and white if the problem was skipped over.

Problem Page

The Project Page, which runs in the main tab, can be used to create a problem tab containing the Problem Page for any one of your problems. The Problem Page performs common operations for a problem. At any given time a problem tab contains either the Problem Page, or the Run Page which executes problem submissions and sample runs, or the Option Page which can be used to edit execution options.

You can have several problem tabs at one time. However two different problem tabs must be for different problems.

The Problem and Run Pages in a problem tab may run background jobs that take a while to execute, and while a background job is running for a problem, you cannot do anything else with that problem, except click the `**ABORT**' button to abort the background job. But you can work on a DIFFERENT problem, which will be in a different problem tab.

Problem Names

A **problem name** can contain only letters, digits, and the special character underline(_). In addition, it must begin with a letter and end with a letter or digit. Unlike other EPM names, a problem name can NOT contain a dash(-), as problem names must also be JAVA class names.

Problem Marks

On the Problem Page:

- ↓ means `show below'
- ↑ means `hide' (don't `show below')
- X means `toggle overstrike of file name'; overstuck files are deleted when you `Delete Over-Struck Files' or upload or make a new file or make a new link
- ⇒ means `make a new file with given extension from the file named at the beginning of the line'

Current Problem Files

The **current problem files** are the files that you have uploaded or made for the problem, plus links to other files. These files and links are stored in your problem directory and can be viewed, used, or deleted. They are displayed in the `Current Problem File' table, one line per file or link.

The order in which the file lines appear can be changed. The default is `**extension order**' which sorts on file extension first and file basename second. You can also have the file lines sorted in `**most recent first**' or `**alphabetic order**' (i.e., basename first, extension second).

Current problem **file names** must have particular extensions that specify their role within the problem. The rest of a file name can contain only letters, digits, and the special characters dash(-) and underline(_), must begin with a letter or digit, and must end with the problem name, which if it is preceded by anything must be preceded by a dash(-).

Link names follow the same rules as file names, and furthermore each link name has the same extension as the file to which it is linked. For example, files imported from the parent of a problem are actually links in your problem directory to the actual file in the parent directory. Usually links are treated and talked about as if they were files.

Clicking on a file name, if it is a button, will show the file with line numbers in a separate auxiliary window. There is an exception for PDF files, which will be rendered as a document in the separate auxiliary window.

Files that are a single short line cannot be shown but their contents are displayed in { } brackets at the end of the file's line.

Empty files and binary files cannot be shown but their nature is displayed at the end of the file's line.

Displayable files must be encoded in UTF-8 or PDF. A UTF-8 file that is displayable has its number of lines indicated at the end of the file's line.

If a file has been pushed to or pulled from a project, the file itself is not in your problem directory, but is in the project's problem directory and is linked to your problem directory. These are marked by `(Link to PROJECT-NAME project)' at the end of the file's line. Files that are links to default EPM system files (such as epm_default_generate) are marked as `(Link to default)'. Files that are links to other files in your problem directory are marked as `(Link to FILE-NAME)'. This last only happens when you provide multiple solutions to a problem: see Multiple Solutions for details.

You can download a file by clicking on its name while holding a control key down, provided the file name is in a button or is in a box with a black boundary line.

File Extensions

EPM files have **extensions** that indicate their format and purpose. In the following PPPP denotes the problem name and XXXX denotes some other part of the name: for example, the name XXXX-PPPP.in matches the name `00-001-reverser.in' if XXXX is 00-001 and PPPP is `reverser'.

When a problem is pulled from a project, many of these files are supplied by the problem parent. The pull may make links in your problem directory to these files in the parent directory, or the commands that make files from other files may read these files directly from the parent directory.

The general scheme for producing a XXXX-PPPP.score file containing a score from a XXXX-PPPP.in test case input file and an executable solution PPPP is:

```
generate-PPPP <XXXX-PPPP.in >XXXX-PPPP.sin
PPPP <XXXX-PPPP.sin >XXXX-PPPP.sout
```

```
filter-PPPP <XXXX-PPPP.sin 3<XXXX-PPPP.sout >XXXX-PPPP.fout
epm_score XXXX-PPPP.fout XXXX-PPPP.ftest >XXXX-PPPP.score
```

The generate-PPPP program takes the .in test case input file and produces the .sin solution input file that is input to the solution program **PPPP** to produce the .sout solution output file. Then both the solution output and solution input file are input to the filter-PPPP program to produce the .fout filtered solution output. Lastly the **epm_score** program compares the .fout filtered solution output to the .ftest test filtered solution output provided by the judge to produce the .score file. The actual commands used to execute all this are more complex because the generate, solution, and filter programs are run in sandboxes, as they may be uploaded. The solution program is of course provided by the problem solver, but the generate and filter programs are provided by the problem setter (the person who creates the problem).

Problem Solution Files

The following files are relevant to solving the problem named PPPP. Some files are supplied by the parent from which the problem was pulled.

When a file is uploaded, another file is made, but if this second file cannot be made successfully, the file to be uploaded is not uploaded. In this case information concerning the error will be available in the Working Files.

Files That Document The Problem:

PPPP.pdf	Problem Description; supplied by parent
XXXX-PPPP.txt	Supplementary data/documentation text file; supplied by parent; see PPPP.pdf file for more information

Supporting Programs

generate-PPPP	Generate Program executable; supplied by parent or linked to epm_default_generator
filter-PPPP	Filter Program executable; supplied by parent or linked to epm_default_filter
display-PPPP	Display Program executable; supplied by parent only if displays are supported for the problem

Extra Documentation Usually Not Needed:

generate-PPPP.txt	Documentation of generate-PPPP program; made from generate-PPPP
filter-PPPP.txt	Documentation of filter-PPPP program; made from filter-PPPP
display-PPPP.txt	Documentation of display-PPPP program; made from display-PPPP if that exists

Solution Source Files One of Which You Upload:

PPPP.c	C Program Source File
PPPP.cc	C++ Program Source File
PPPP.py	PYTHON Program Source File
PPPP.java	JAVA Program Source File

Solution Binary Files Made When Solution Uploaded:

PPPP	C/C++ Program Binary File; made from PPPP.c or PPPP.cc file
PPPP.pyc	PYTHON Program Binary File; made from PPPP.py file
PPPP.jar	JAVA Program Binary File; made from PPPP.java file

Test Files Supplied by Parent

XXXX-PPPP.in	Test Case Input File supplied by parent; for sample tests, XXXX begins with `00-'

XXXX-PPPP.ftest	Test Case Output File supplied by parent; used to make XXXX-PPPP.score
sample-PPPP.run	List of 00-XXXX-PPPP.in Files supplied by parent
submit-PPPP.run	List of XXXX-PPPP.in Files supplied by parent; only appears on Run Page; used to submit problem for judging; non-sample XXXX-PPPP.in files are not on Problem Page unless they had an error during submit

Files Made From Other Files (Not During Upload):

XXXX-PPPP.sin	Solution Input File, made from XXXX-PPPP.in test case input file by generate-PPPP
XXXX-PPPP.sout	Solution Output File, made from XXXX-PPPP.sin file by compiled solution
XXXX-PPPP.dout	Debugging Output File, made from XXXX-PPPP.sin file by the compiled solution and, for C/C++, the solution source code; designed to identify the line number of the statement where a CPU time limit signal or program fault occurred
XXXX-PPPP.fout	Filtered Solution Output File, made from XXXX-PPPP.sout file and XXXX-PPPP.sin file by filter-PPPP
XXXX-PPPP.ftest	Filtered Solution Test File, made by simply copying the XXXX-PPPP.fout file (WARNING: affects scoring; forces `Completely Correct' score);
XXXX-PPPP.score	Scoring Output File, made from XXXX-PPPP.fout and XXXX-PPPP.ftest files by scoring program
XXXX-PPPP.pdf	Solution Output Display File, made from XXXX-PPPP.in by generate-PPPP, compiled solution, display-PPPP, and epm_display
XXXX-PPPP.rout	Run output file, made from XXXX-PPPP.run and other files by the scoring and other programs; summarizes .score files made from .in files listed in the .run file
XXXX-PPPP.txt	Documentation of the XXXX-PPPP program (for XXXX equal `generate', `filter', or `monitor') made by executing `XXXX-PPPP -doc'

Working Files That Should Be Empty:

PPPP.cout	Compiler Output File, made during upload or debug compilation
PPPP.cerr	Error Output from compilation during upload or debug compilation
XXXX-PPPP.d1err	Error Output from display-PPPP
XXXX-PPPP.d2err	Error Output from epm_display
XXXX-PPPP.ferr	Error Output from filter-PPPP
XXXX-PPPP.gYerr	Error Output from generate-PPPP (if generate-PPPP is run twice Y is 1 or 2, else Y is missing)
XXXX-PPPP.rerr	Error Output from a run
XXXX-PPPP.serr	Error Output from solution program
XXXX-PPPP.scerr	Error Output from epm_score scoring program

Working Files That Are Not Empty:

XXXX-PPPP.disp	Output from display-PPPP; made by display-PPPP if and when XXXX-PPPP.pdf is made

Test Files You May Uploaded:

XXXX-PPPP.in	Extra Test Case Input File; XXXX-PPPP.sout file is made by upload
XXXX-PPPP.run	Extra List of .in Files

Test Files Made During Optional Uploads:

XXXX-PPPP.sout	Solution Output File, made from XXXX-PPPP.in file by generate-PPPP and compiled solution

Problem Creation Files

The following files are relevant to creating the problem named PPPP. These are in addition to the files listed above in Problem Solution Files.

Files That Must Be Uploaded:

One Solution Source File

PPPP.tex	LATEX Document Source File
XXXX-PPPP.in	Test Case Input File; one of many; if XXXX begins with `00-' this is a sample input file; otherwise it is a submit (judge's) input file; the solution source and optional generate-PPPP.cc must be uploaded before .in files are uploaded
sample-PPPP.run	List of sample .in Files; the optional generate-PPPP.cc, the .in files listed, and the optional filter-PPPP.cc must be uploaded, and the .ftest files associated with the listed .in file must be made, before a .run file is uploaded
submit-PPPP.run	List of submit .in Files

Files That May Be Uploaded:

generate-PPPP.cc	C++ generate-PPPP Program Source File; upload before any .in files
filter-PPPP.cc	C++ filter-PPPP Program Source File; upload before making .ftest files, hence before uploading .run files
display-PPPP.cc	C++ display-PPPP Program Source File
XXXX-PPPP.txt	Supplementary Data/Documentation Text File

Files Made During Upload:

PPPP.pdf	Document PDF File, made from PPPP.tex file

One Solution Binary File

XXXX-PPPP.sout	Solution Output File; made from XXXX-PPPP.in file by generate-PPPP and compiled solution
generate-PPPP	C/C++ Program Binary File; made from generate-PPPP.c or generate-PPPP.cc file
filter-PPPP	C/C++ Program Binary File; made from filter-PPPP.c or filter-PPPP.cc file
display-PPPP	C/C++ Program Binary File; made from display-PPPP.c or display-PPPP.cc file

Files Created When Making Other Files:

generate-PPPP	Generate Program executable; created automatically by linking to epm_default_generator if no generate-PPPP.cc uploaded
filter-PPPP	Filter Program executable; created automatically by linking to epm_default_filter if no filter-PPPP.cc uploaded

Files Made From Other Files (Not During Upload):

XXXX-PPPP.fout	Filtered Solution Output File, made from XXXX-PPPP.sout file and XXXX-PPPP.sin file by filter-PPPP
XXXX-PPPP.ftest	Filtered Solution Test File, made by simply copying the XXXX-PPPP.fout file (WARNING: affects scoring)
XXXX-PPPP.rout	Run output file, made from XXXX-PPPP.run and other files by the scoring and other programs; summarizes .score files made from .in files listed in the .run file

Working Files That Are Not Empty:

PPPP.tout	LATEX output file made from PPPP.tex file
PPPP.log	Log file of a LATEX compilation, made from PPPP.tex file; like .tout file but with extra information
PPPP.fls	File List that lists all input and output files in a LATEX compilation, made from PPPP.tex file

Generate and Filter Programs

The generate, filter, monitor, and scoring programs all follow the conventions that a line beginning `!!' is special, that input (.in) file lines beginning with `!!##' are comment lines and are removed (by the generate program) so the solution never sees them, and that solution output (.sout) file lines beginning with `!!**' are comment lines that may be used for debugging output (and are ignored by the scoring program).

Other lines beginning with `!!' should only be used in .in input files with generate programs that understand how to interpret them.

The default generate program simply copies its input to its output removing lines beginning `!!##', which are considered comment lines. If you are writing your own problem, you may write a generate program (e.g., generate-PPPP.cc) that treats lines beginning with `!!' as instructions to generate a sequence of input lines, typically using a pseudo-random number generated built into the generate program (so it will not change when standard libraries are updated) and seeded by a number in the input instruction line.

For example, `!!L seed m n' could generate a line containing m numbers randomly chosen in the range [1,n] with seed being the pseudo-random number generator seed. Here `L' is the literal letter L, and seed, m, and n are natural number parameters. See the Downloads Page (accessible from any Problem Page) for source code of an example generate program.

The default filter program simply copies solution output (which it gets from file descriptor 3) to its standard output. The solution output can contain comments beginning `!!**'. If you are writing your own problem, you may write a filter program (e.g., filter-PPPP.cc) that computes data from the solution input (output of the generate program) and the solution output and replaces part of the solution output with lines it produces.

For example, if the solution is supposed to produce a shortest path in a graph, then the filter might check that the answer is in fact a graph path, and replace the answer by the length of the path. The filter reads the solution input from its standard input, file descriptor 0, to construct the graph, and reads the solution output from file descriptor 3. See the Downloads Page (accessible from any Problem Page) for source code of an example filter program.

The epm_score scoring program treats any line beginning with `!!' as a comment line, which it ignores. If the default generate program sees a line beginning with `!!' but NOT `!!##', it treats it as a comment line and removes it, but also issues a warning message on to its standard error file. Similarly if the default filter program sees a solution output line beginning with `!!' but NOT `!!**', it outputs the line but issues a warning message.

The Display Program

The display-PPPP program reads an XXXX-PPPP.sin file and its corresponding XXXX-PPPP.sout file and produces an XXXX-PPPP.disp file that can be input to the epm_display program to produce a XXXX-PPPP.pdf file containing a geometric display. The scheme for doing this is:

```
generate-PPPP <XXXX-PPPP.in >XXXX-PPPP.sin
PPPP <XXXX-PPPP.sin >XXXX-PPPP.sout
display-PPPP <XXXX-PPPP.sin 3<XXXX-PPPP.sout >XXXX-PPPP.disp
epm_display <XXXX-PPPP.disp >XXXX-PPPP.pdf
```

Note the display-PPPP gets its .sin file from file descriptor 1 and its .sout file from file descriptor 3. The input/output structure of display-PPPP is the same as that of a filter-PPPP program, so if you are writing display-PPPP.cc you may find the filter template epm_filter.cc useful.

There are no general standards for formatting the input to display-PPPP; the input formats are completely problem-specific.

Multiple Solutions

If you are writing in the C language, so that normally you would provide a solution PPPP.c, you may provide multiple solutions named YYYY-PPPP.c, for various YYYY. Then after upload you will have files YYYY-PPPP and YYYY-PPPP.c with `Link' buttons after them. Clicking a `Link' button for YYYY-PPPP will create a link PPPP → YYYY-PPPP, and similarly the `Link' button for for YYYY-PPPP.c will create a link PPPP.c → YYYY-PPPP.c. This last is only useful if you are making something that requires the source file, such as a .dout debugging file.

Other source languages behave similarly.

When a link is being created to an executable solution file, all executable solution files are first deleted automatically. This is a convenience feature, as if you have several executable solutions for different programming languages, you cannot make .score files and other files made with solution executables. However this does not apply to solution source files: e.g., you can have PPPP.c and PPPP.java at the same time.

When choosing names YYYY-PPPP.c for your solutions, you must avoid having YYYY be any of `generate', `filter', or `monitor' or end in `-generate', `-filter', or `-monitor', as such files are treated not as solutions but as generate, filter, or monitor programs.

Commands Last Executed

This section displays the UNIX/POSIX/LINUX commands last executed to make a file, and if any files output by these commands are kept in the problem directory, a list of kept files is also displayed.

Included is the CPU execution time in 0.001 second units of some of the command lines.

If a command line fails, this is indicated along with a failure message (but not for debugging runs that make .dout files which contain their own failure messages, if any). The indications take the form of footnotes marked by *, **, ***, etc.

The lines of kept files in the Current Problem Files section are also highlighted by using a darker background color. Some of these files, particularly PDF files, are automatically shown in a separate auxiliary window.

The `**Delete Working Files and Command History**' button deletes this section and stops any kept files from being highlighted or automatically shown.

Working Files of Last Executed Commands

This section lists the files made by the last executed commands which were not kept. These files can be displayed in the same manner as the files in the Current Problem Files section.

Working files that should be examined after an unsuccessful run are highlighted in yellow. Some of these files may be automatically displayed.

Files that are kept on successful uploads or makes will remain as (usually non-empty) working files on unsuccessful uploads or makes.

Working files are deleted when a new set of commands is executed, when a .run file is run, or when the `**Delete Working Files and Command History**' button is clicked.

Run Page

If you click a **Run** button on the Problem Page, the associated .run file will be run and the page will switch to the Run Page where the results will be displayed.

The Run Page displays both .run files that are listed on the Problem Page, and may be `Run', and .run files that are only in the parent of the current problem, and may only be `Submit'ed.

A **.run file** is just a list of .in files from which .score files may be computed. A .run file can be uploaded on the Problem Page, and pushed along with other files to a project by the Project Page. When a project is pulled, most .run files are left in the problem's parent and can only be `submitted', but sample run files are linked from the problem's own directory to the parent, and can only be `run'. The exact differences between a run and a submit are described at the end of this section.

Both runs and submits are referred to as `run's. A run produces for each .in file an associated .score file. During the run, and after it is over, the Run Page displays for each .in/.score pair a line giving the .in/.score files' common base name and the CPU execution times for the various sub-computations needed to compute the .score file from the .in file. These times are marked as follows:

g1: generate program making input for solution
s: solution making .sout file from generated input
g2: generate program making input for filter program
f: filter program making .fout file from .sout file and generated input

The Run Page lists all the .run files in a table, one file per line. For the last run of a particular .run file, the associated run output, or .rout file is given. If there is also a non-empty run error, or .rerr file, it is also listed. These files can be displayed in an auxiliary window by clicking on their name, and can be downloaded by holding the control key down while clicking on their name.

The objective of run is to produce a `Score' for the total run that is listed in the run's output .rout file. Sometimes no score can be produced, and none is listed. The most frequent cause of this is failure to upload a problem solution on the Problem Page before attempting the run.

The differences between a `Run' of a a .run file in your directory and a `Submit' of a .run file in the parent of your directory are as follows. During a submit, all files used are in the parent, and not your local directory, except for your solution executable. During a run, files in your local directory override those in the parent. After a submit that produces a `Score', the .rout file is recorded as a submit result in the parent (project) problem directory, where every user can view it (using the View Page). The .rout file is also recorded in your problem directory for a submit that produces a `Score'. After a non-submit run that produces a `Score', the .rout file is only recorded in your problem directory where only you can view it.

Lastly, after a submit that produces a `Score' that is not `Completely Correct', .in and .ftest files of the first failed test case will be linked into your problem directory so you can debug against them without re-submitting, provided that you have `first-failed' privilege for the project problem.

Option Page

The **Option Page** shows the options given to commands that perform compilations and computations, and allows you to edit these options within limits.

For example, the allowed maximum size of the virtual memory when a problem solution program is executing is an option, as is the allowed maximum size of any file output by the solution.

There are two kinds of options: numbers and arguments. Numbers are substituted into arguments, and arguments are substituted into the commands used to make one file from another file. These commands are specified by templates, which can be displayed by clicking the `View Templates' button to display the Template Page in a separate auxiliary window.

There are two values for each option: an **inherited value** and a **local value**. The inherited value is set by the parent of the current problem, and cannot be changed (without pushing a new options files to the parent). The local value, when given, overrides the inherited value. In displays, local values are colored green and inherited values are colored tan. Note that parent .run file submissions use only inherited option values, whereas current problem .run file runs use local overrides.

As with other pages, clicking on a ↑ will hide a section of the page and clicking on a ↓ will display the section.

You can switch back and forth between just viewing options and editing them by clicking buttons at the top of the Option Page. When editing you can also reset all options to their inherited values.

Number Options

Number options are numbers included in argument options. For example, the STIME option value is the maximum number of CPU seconds allowed a C/C++ problem solution program, whereas SJTIME is the maximum allowed a JAVA solution program, and SPTIME is the maximum allowed a PYTHON program.

If you edit an inherited number option, its value will turn green, even if you input the same value as the inherited value. Clicking the ↻ symbol after the value will restore the value to the inherited value (and restore the color to tan).

Argument Options

Argument options are the arguments that are inserted into template commands. For each such option there are discrete choices (for some argument options there is no choice and the single value is `required').

Option numbers are inserted into argument options. For example, the STIME number option is inserted into the `-cputime STIME' value of the SSTIME argument option. The SSTIME argument in turn is inserted into the template commands that run a C/C++ solution program, so that the options work together with the commands to determine the maximum number of CPU seconds allowed a C/C++ solution program.

The absence of any value for an option command argument is represented by an empty box. If there are two colored boxes, the one colored green is the current value and is local, and the one colored tan is inherited. If only one box is colored, the option value is inherited, even if the box is colored green (it is green because you are editing and have clicked on the inherited value).

Template Page

A **Template** supplies the commands used to make one file from another file. Templates can be displayed in a separate auxiliary window by clicking the `View Templates' button on the Option Page.

An index of all the available templates is displayed at the top of the page. Clicking on the box next to a template name displays or hides the relevant contents of the template.

The commands displayed for a template are executed when the template is used. These contain argument names substituted from the argument options. Each substitutable argument name is preceded in its command by a dollar sign ($). In addition to argument option substitutions, the problem name is substituted for PPPP, and the remainder of the source file name is substituted for other sequences of 4 consecutive capital letters, such as XXXX or YYYY.

The template selected for a particular task is controlled by the extensions of the input and output files, but also by what other files are available. For example, several templates are available to make a .score file with a score from an .in file with solution input. One of these is selected if there is a binary file with no extension (for C or C++); another is selected if there is a binary file with the .jar extension (for JAVA); and a third is selected if there is a binary file with the .pyc extension (for PYTHON). Files which are required to use a template are call the **required files** of the template.

A template allows some required files to be **creatable**. These are automatically created from defaults if they do not exist. For example, the generate_PPPP program will be created from a default that does nothing but strip comments from a .in file to make a .sin file.

A template requires some files to be **local**, that is, not in the parent of the current problem. Other files are required to be **remote**, that is, they must be in the parent of the current problem (i.e., in a project). Lastly many required files can be either local or remote. When a creatable file is created, it is always created locally, so it must not be a remote required file.

In addition the template selected is controlled by **conditions**. For example, to be used during the upload of a PPPP.cc file, a template must have the `UPLOAD PPPP.cc' condition. As another example, a template used when a .run file is submitted must have the `SUBMIT' condition. Much of the time only templates with no conditions are selected.

After commands created by a template are executed, the list of **checks** defined by the template are processed. Each check has the form [f1,f2,...] for file names f1, f2, ..., or the check is a single file name f which is equivalent to [f]. For all the checks but the last, the check is ignored if f1 does not exist or has zero size. Generally f1 is an error file. If a check that is not the last is NOT ignored, then `f1 is not empty' is added to the error messages for the execution, and the first two showable files in the list f1, f2, ... are automatically shown.

The last check differs in that it is ignored only if the execution has errors, and otherwise is the list of files to be automatically shown.

Finally the template specifies a **keep** list of files that are to be kept if there are no errors in an execution.

List Page

A problem list is just a list of problems that are either your problems or project problems. The `Your Problems' list is automatically created and lists all of your problems (including those pulled from or pushed to a project). The `public Problems' list is automatically created and lists all problems in the public project. It is an example of a project problem list.

The automatically created lists can get too long to be useful, so you can create shorter lists of your own, and publish them so others can use them.

The list of problem lists can also become too long, so you have a `favorites list' that is a list of the problem lists you are interested in. The favorites list can be edited via the Favorites Page. When you are asked to select a problem list, the list of problem lists presented for selection is always the favorites list.

The favorites list is initialized to include `Your Problems' plus all published problem lists. When a problem list is published (by anyone), it will be automatically added to your favorites list; but you can remove it from your favorites list after it has been automatically added, even if it remains published.

The List Page allows you to view and edit two problem lists (colored tan and blue). Two lists are allowed so you can move problems from one to the other.

To create a problem list of your own, type its name into either of the two `New Problem List Name' boxes on the List Page. When it is created the list is added to the beginning of the favorites list.

You can select either a tan or blue list by creating a new list or selecting a list. Once selected, if the list is your list, you can reorder the list by dragging problems to the location just above where you want to drop them. You can remove problems from the list and un-remove them by clicking the oval next to the problem. The problems with black ovals are in the list, and those with white ovals have been removed (but could be put back in by clicking their ovals).

If you have selected two problem lists, you can drag a problem from one list to the other.

Lists that are not your own cannot be changed. If you drag a problem from such a list to a list of your own, the problem will be copied to the destination list.

If you drag a problem from one of your own lists to another of your own lists, the problem will be moved; that is, it will be removed from its source list while being added to its destination list. If, however, you hold down the control key while dragging the list, it will be copied and not moved; that is, the problem will be left in the source list while being added to the destination list.

After editing a list of your own, you must `SAVE' it. `FINISH' will also save the list and in addition will de-select the list so you can select another list. `CANCEL' will de-select the list without saving it. `RESET' will reset the list to its state before you changed it, and `DELETE' will delete the list completely.

You can also `PUBLISH' an unpublished list and `UNPUBLISH' a published list. A **published** list is visible to other users on their Favorites Page, so these other users can include your list in their favorites. An unpublished list is not visible. When you publish or unpublish a list, the list is saved as by `SAVE', unless the list has not been changed since it was last saved.

When you publish a list, it is automatically added to the favorites list of all users, but they can remove it from their favorites list after it has been automatically added.

List Descriptions

If you have a list of your own named XXXX, then when you have selected this list for editing, you may upload a list description file named XXXX.dsc which contains a description of your list. This description is then visible whenever the list is being considered for addition to someone's favorites list. When you upload a description, the list is saved as by `SAVE'.

You can delete a list description by uploading an empty .dsc file.

Description files contain paragraphs separated by blank lines. There are three kinds of paragraphs: headed, indented, and ordinary. A headed paragraph begins with a `*'; its first input line with the `*' removed is made boldface; and then the paragraph is output as an HTML <p> paragraph with all output lines but the first slightly indented. An indented paragraph begins with a single space or tab and is output in monospace font as

an HTML <pre> paragraph with spaces and line breaks preserved. An ordinary paragraph does not begin with a `*', space, or tab, and is output as a plain HTML <p> paragraph.

A list description file may NOT contain the characters < or >, but may contain &...;' representations of characters; e.g., < for <. For indented paragraphs you may use one or more tabs at the very beginning of a line, but nowhere else. Each tab will be replaced by 8 single space characters.

Favorites Page

The `Favorites' list is a list of those problem lists you find most interesting, in order of interest. Whenever you are asked to select a problem list, it is the favorites list that is given as the list of available problem lists.

This Favorites Page allows you to edit the favorites list. It displays all the lists that can be on the Favorites list, each with an oval that is black for those problem lists that are on the Favorites list. and white for other lists. The black oval lists are in the order that they appear in the Favorites list. Lists created by users may also have an attached description created by their author and displayed just after the list name.

If you click on a oval it toggles between black and white. In either case its list is moved to the boundary between black and white lists, so the black lists are always above the white lists.

You can change the order of lists by dragging a list to a second list below which you want the first list to be. To move a list to the top, drag it to the `Favorites' title line.

When you first alter the favorites list, the `Go To ... Page' buttons are replaced by `SAVE' and `RESET' buttons. Changes you make are not recorded until you click `SAVE'. Clicking `RESET' restores Favorites to its previous state. You must use one of these two buttons to get the `Go To ... Page' buttons to reappear.

View Page

The View Page allows you to see lists of actions that involve an account, a project, a problem, or the published lists.

To view actions involving an account select the account.

To view actions involving a project select the project.

To view actions involving a problem, first select a problem list, and then select a problem from that list.

The actions are displayed as action lists. The actions are generally self explanatory. For the submit actions, the 3 decimal digit number followed immediately by `s' is the maximum solution execution time in seconds.

You can filter the actions displayed by entering a logical expression which must be satisfied by each action displayed. This logical expression uses keywords in the action, which are pretty much any significant word in the action. The logical expression syntax is:

logical expression ::= term { whitespace term }*

term ::= factor { '&' factor }*

factor ::= keyword | '-' keyword

A keyword factor is true iff the keyword is in the action. A `-keyword' factor is true iff the keyword is NOT in the action. & denotes logical AND, and whitespace denotes logical OR. There are NO parentheses. Note that terms cannot contain whitespace.

Projects for which you do not have view privilege will NOT be visible on the View Page. Such projects are usually used to prepare programming contest problems.

Manage Page

The Manage Page allows you to view project and project problem privileges, to change these if you have owner privilege for a project or project problem. The Manage Page also allows you to download gzipped .tgz tar files of problems and projects if you have download privileges, and to move problems between projects if you have move privileges.

On the Manage Page you begin by selecting either a project or a problem.

When you select a project, the privilege file of the project is displayed, and if you have owner privilege for the project, you can edit this file. A button also appears that allows you to to download a .tgz tar file of the project contents if you have download privileges for the project.

When you select a project (non-local) problem, the privilege file of the project problem is displayed, and if you have owner privilege for the project problem, you can edit this file. A button also appears that allows you to download a .tgz tar file of the problem's directory within its project, if you have download privileges for the project problem. A selector also appears which allows you to select a target project to move the problem to, if you have move-from privileges for the problem's current project (you must also have move-to privileges for the target project).

When you select a non-project problem, a button appears that allows you to download a .tgz tar file of the problem's local directory (you need no privileges for this).

The following are the possible project privileges:

owner	allows user to edit the privileges of the project
push-new	allows problems to be pushed to the project if they are not already in the project
pull-new	allows a problem in the project to be pulled if the user does not already have a problem of the same name, or if the user's problem of the same name has no parent
re-pull	allows a problem in the project to be pulled when an existing user problem has the project problem as its parent
view	allows project actions to be viewed in the View Page
show	makes the project problems visible; e.g., allows the project problems list to be included in the user's Favorites and if this is done, allows project problem descriptions to be displayed via the Projects Page
move-from	allows problems to be moved from the project to other projects
move-to	allows problems to be moved to the project from other projects
download	allows a .tgz tar file to be made containing the contents of the project directory
first-failed	allows the input and output of the first failed test case of a project problem submission to be linked into the submitter's problem directory

The following are the possible project problem privileges:

owner	allows user to edit the privileges of the project problem
re-push	allows user to push a user problem which has this project problem as its parent
download	allows a .tgz tar file to be made containing the contents of the problem subdirectory of the project directory
first-failed	allows the input and output of the first failed test case of a submission of the problem to be linked into the submitter's problem directory

You do not need privileges to download a .tgz tar file containing the local directory contents for a problem.

To determine privileges, lines in privilege files are processed in the order

1. the $epm_priv_prefix EPM server parameter lines (the parameter value is processed as if it were the contents of a file)
2. project privilege file lines
3. project problem privilege file lines, if the privileges of a project problem are being determined (as opposed to the privileges of just a project)

The $epm_priv_prefix parameter is used to grant owner privileges to EPM server administrators.

A privilege file line is one of the following:

- blank line; ignored
- comment line; first non-whitespace character is `#', ignored
- line of form `<sign> <key> <matcher>'
  where
  - <sign> ::= `+' | `-'
  - <key> ::= <privilege> | <group>
  - <matcher> ::= <regular-expression> | <group>
  - <privilege> ::= `owner' | ...
    where <privilege> must be appropriate for the file it is in (e.g., `re-push' is appropriate for a project problem file but not a project file)
  - <group> ::=
    sequence of letters, digits, `_', `-', and `@' which begins with a `@' followed by a letter, has only one `@' (at its beginning), and ends with a letter or digit

  If this line matches, the line assigns <sign> to <key> unless <key> has previously been assigned.
  A <regular-expression> matches if it matches the current user ID.
  A <group> matches if it has previously been assigned `+'.

If after all the privilege file lines are processed a privilege has the value `+', the current user has the privilege. Otherwise the current user does NOT have the privilege.

Groups may be defined that are sets of users, or are intersections, unions, or set complements of previous groups.

Conflicts

Browser Back, Forward, and Refresh Buttons Must NOT Be Used
: You must not use the browser back, forward, and refresh buttons. You can refresh using control-R or the refresh button provided ON THE PAGE, in the upper right corner. Refreshing may change the state of the page.

One Session Per Browser
: A browser cannot have more than one session at a time with a particular EPM server. To kill the session so you can start a new session with a different login, you must either terminate and re-start the browser, or go to first the User Page and then on to the Logout Page to log out.

Multiple Sessions
: If you are logged in on one browser and you log in on a second browser, your first login session will be aborted. This is a form of automatic logout. You should get a message when you try to use the aborted session. The purpose of this is to prevent two sessions for the same user stepping on each others toes.

IP Address Changes
: If your browser changes its IP address while you are logged in, your session will become unusable, unless the server is secure (uses SSL and certificates) and has been set to not check IP addresses.

Multiple Logins (Main Tabs) in the Same Session
: If you log in using a fresh browser tab and you are already logged using another tab on the same browser, your old login main tab will become orphaned and unusable. Only the old login main tab is

affected, not the session or problem tabs.

Multiple Tabs for the Same Problem
: You should NOT open a problem tab by using a URL in a new tab. If you do and there is already another tab for the same problem, this other tab will become orphaned and unusable. Always use the Project Page selector to create a problem window.

Deficiencies

Moving Problems Between Projects
: To be implemented on Manage Page. Not yet implemented.

New Project Creation
: The Manage Page does not allow users to create projects.

Score Board
: There is not yet any scoreboard. Submissions shown on the View Page provide data sufficient to construct a scoreboard.

No Automatic Contest Start
: There is not yet a system for automatically starting a contest by resetting its project's privileges at a specified time.

Times Not Explained
: Times attached to problem in problem lists and to list in the favorites list are not explained, and might not be that useful.

Per Guest Privileges
: A user guest entry is to have extra information after the guest name that restricts which problems the guest can see. Not yet implemented.

JAVA Output File Too Large
: This is not correctly scored.