Project Number: NTH – AAWH

Student Progress in ASSISTments

A Major Qualifying Project Report:

Submitted on May 5, 2016

To the Faculty of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the Degree of Bachelor of Science

By:

Patrick Desmarais

Advised by Professor Neil Heffernan

Abstract

This project extends the ASSISTments tutoring system by incorporating more immediate feedback into the existing system. The goal was to create a way to more accurately provide students with an idea where in a problem set they are and how many more problems they can expect before completion. To accomplish this goal, a new infrastructure was implemented to differentiate between different types of child problem sets inside of a parent problem set to accurately create sections that would appear in the Progress Panel providing the student with the information about their progress.

Acknowledgements

I would like to thank Professor Heffernan not only for his support on this project but also for his support throughout my time at WPI. I would also like to thank David Magid and Chris Donnelly for their patience and assistance this year. Finally I would like to thank the entire ASSISTments team for their help and for making my time in the ASSISTments lab extremely enjoyable.

Contents

Abstract 1
Acknowledgements 2
Contents 3
Table of Figures 4
Introduction 5
Background 6
    Problem Set Types 6
Complete All: 6
Skill Builder: 6
Choose One: 6
If-Then-Else: 7
Random: 7
    Progress Panel 7
    Motivation for Change 9
Methodology 11
    Requirements 11
    Progress Panel Separation 11
    Design of New Implementation 12
    Results 15
    Future Work 22
    Conclusion 24
    Appendix 25
Source Code 25

Table of Figures

Figure 1: Previous Progress Panel 7
Figure 2: Problem Set Structure 8
Figure 3: Previous Progress Panel Finished Problem Set 9
Figure 4: Starting Problem Set 15
Figure 5: First Skill Builder Section 16
Figure 6: Section of Two Single Problems 17
Figure 7: Second Skill Builder Section 18
Figure 8: Linear Problem Set Section 19
Figure 9: Last Problem 20
Figure 10: Problem Set Completed 21

Introduction

ASSISTments is an online tutoring tool that has been in development at Worcester Polytechnic Institute since 2003. The goal of ASSISTments is to provide immediate feedback to students, teachers, school administrators, and parents. Teachers can utilize pre-existing problem sets, edit existing problem sets, or create their own problem sets which they can then assign to their students to complete. When given an assignment on paper or problems to be completed from a book a student can see how many problems they must complete right away. Inside of ASSISTments however, there was no way for a student to know how many problems to expect from an assignment outside of a teacher telling the students. The focus of this project was to update the visual display a student sees when completing a problem set with information that would provide them with an idea of how close they are to completion and to show what they have already completed in a better way.

Background

This section of the paper provides information that was critical to understand before beginning the project. First the different kinds of problem sets available to teachers will be explained. Then I show how the user interface appeared and detail its shortcomings before the changes made from this project.

Problem Set Types

Complete All:

The most basic problem set type. Here the student must complete each problem and they are unable to progress from problem to problem until they get the correct answer.

Skill Builder:

A Skill Builder requires that a student get n-right-in-a-row. This number is usually 3 but can be changed by the teacher or content creator. These types of problem sets are ideal for helping a student learn and master a concept.

Choose One:

In a Choose One problem set a student is randomly placed into one of the sections created in the problem set by the system. This placement is not exposed to the student and they have no idea they were placed into a condition. This type of problem set is ideal for conducting a study.

If-Then-Else:

An If-Then-Else problem set is structured very similarly to if statements in many programming languages. The first is the “if” or conditional. Here a student completes a problem or section and based on their performance is placed in either the “then” or “else” part. This can be used to set up problem sets where a student’s performance decides what if any future work they do for that problem set.

Random:

It is important to note that both Complete All and Skill Builder problem sets can be assigned in a linear order (a student progresses through problems in the order they are set up in the problem set) or in random order.

Progress Panel

The Progress Panel is the name given to the left most area in the tutor that a student sees when completing a problem set. This is the area located within the red box in Figure 1 below.

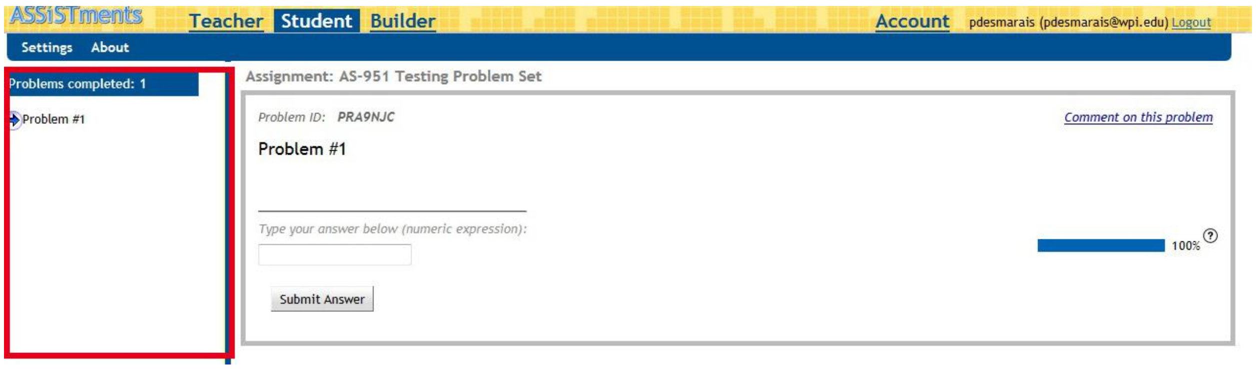
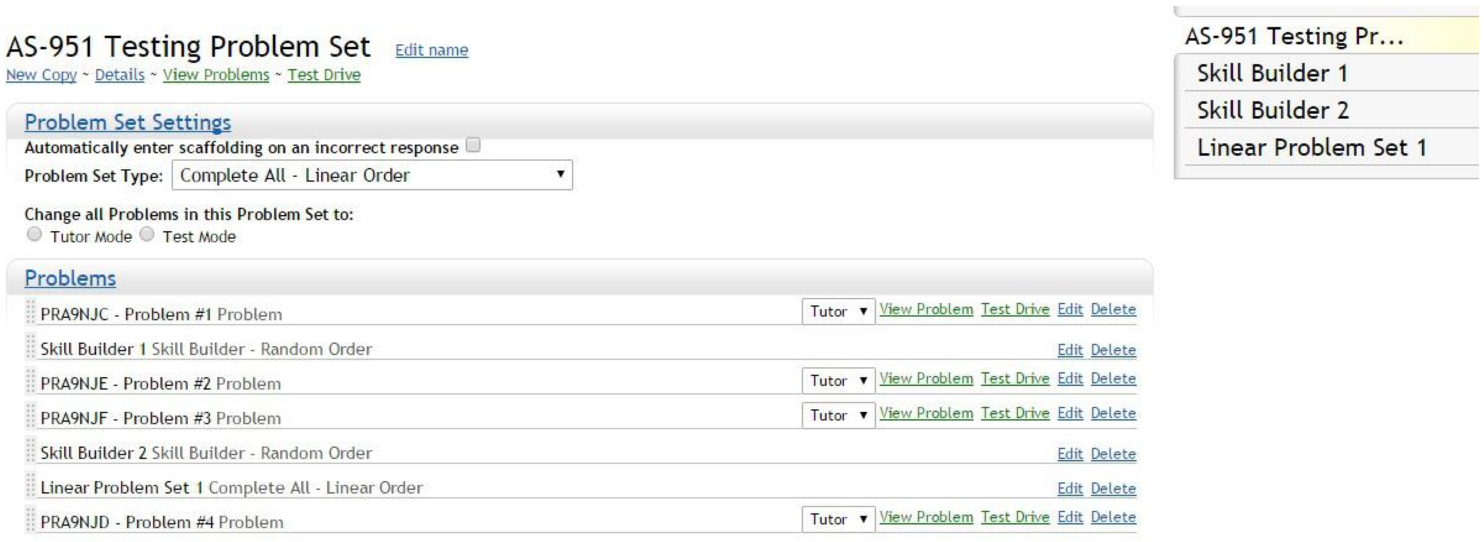


Figure 1: Previous Progress Panel

The area to the right of the red box is where the current problem is displayed to the student. As the student completes problems the Progress Panel is updated with each problem, how the student did on that problem, and the student's overall progress on the assignment. If it is a simple Complete All then the header at the top will display the number of problems completed/the number of problems in total. The header of a Skill Builder by itself will show the student that they need to get n right in a row. The problem arises when a problem set (the parent problem set) contains children problem sets of various kinds. Figure 4 shows the builder view of a problem set that showcases this limitation.



AS-951 Testing Problem Set Edit name
New Copy ~ Details ~ View Problems ~ Test Drive

Problem Set Settings
Automatically enter scaffolding on an incorrect response ☐
Problem Set Type: Complete All - Linear Order

Change all Problems in this Problem Set to:
○ Tutor Mode ○ Test Mode

Problems

PRA9NJC - Problem #1 Problem	Tutor	View Problem Test Drive Edit Delete
Skill Builder 1 Skill Builder - Random Order		Edit Delete
PRA9NJE - Problem #2 Problem	Tutor	View Problem Test Drive Edit Delete
PRA9NJF - Problem #3 Problem	Tutor	View Problem Test Drive Edit Delete
Skill Builder 2 Skill Builder - Random Order		Edit Delete
Linear Problem Set 1 Complete All - Linear Order		Edit Delete
PRA9NJD - Problem #4 Problem	Tutor	View Problem Test Drive Edit Delete

AS-951 Testing Pr...
Skill Builder 1
Skill Builder 2
Linear Problem Set 1

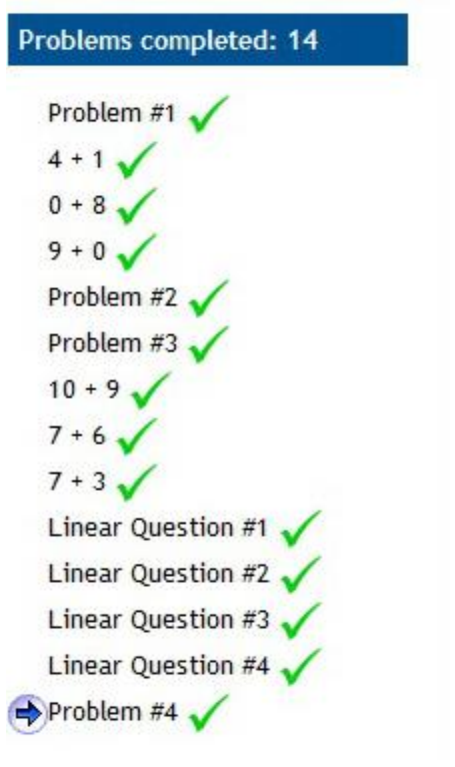
Figure 2: Problem Set Structure

This problem set contains multiple child problem sets. The breakdown is like this:

1) Single problem
2) Skill Builder problem set
3) Single problem
4) Single problem
5) Second Skill Builder problem set

6) Linear Complete All problem set
7) Single Problem

Under the previous implementation the system had no way to provide an exact number of problems left for the students to complete when problem set types were together. In these cases the Progress Panel would simply show the total number of problems completed so far. Figure 5 shows the Progress Panel at the completion of the previously mentioned problem set in the old implementation.



Problems completed: 14

Problem #1 ✓
4 + 1 ✓
0 + 8 ✓
9 + 0 ✓
Problem #2 ✓
Problem #3 ✓
10 + 9 ✓
7 + 6 ✓
7 + 3 ✓
Linear Question #1 ✓
Linear Question #2 ✓
Linear Question #3 ✓
Linear Question #4 ✓
Problem #4 ✓

Figure 3: Previous Progress Panel Finished Problem Set

Motivation for Change

As previously stated the main goal of ASSISTments is to provide immediate feedback. Under the old Progress Panel implementation a student had no idea how far along in a problem set they were. Updating the Progress Panel to display each child problem set or section to a student they can have a better idea of their progress as they complete a problem set. In addition this change also keeps with the fundamental philosophy of ASSISTments in providing immediate feedback.

Methodology

This section details the process of updating the Progress Panel. I will first describe the requirements that the updated panel should fulfill. Then the separation of the Progress Panel into its own presenter will be discussed. Finally an overview of the new Progress Panel implementation will be given.

Requirements

There were three main requirements to be met in the new implementation. First was what information the new headers would display based on the problem set type. The top header would display the total count of problems completed and start at zero. The headers of each section would then show one of the following depending on type:

1) Total Problems completed: n
2) Problems completed: n
3) Problems completed: x/n

The second requirement was an indicator to the student if there is more to complete after the section they are currently working on. This indicator will be referred to as the "More to come…" label. The last requirement was to take the code for the Progress Panel and move it to its own class. This will be discussed in the next section.

Progress Panel Separation

Before the Progress Panel could be updated with new functionality, it first had to be moved into its own class. Previously the code for the Progress Panel lived inside of another class called the ApplicationPresenter. Anything that needs to be displayed is handled by a Presenter class. The ApplicationPresenter is in essence the main

Presenter that gets everything going. With this structure of everything having its own Presenter class it felt wrong to have the Progress Panel live inside of another Presenter. The first step was to create a new Presenter class for the Progress Panel, called ProgressPanelPresenter, to handle the actions on that panel and to remove the remnants from the ApplicationPresenter. In the new implementation the ApplicationPresenter no longer listens for actions by the user (student), now handled by the ProgressPanelPresenter.

This was an important exercise for a couple of reasons. The first reason is moving the code into its own class makes the code both cleaner and easier to read. Furthermore the separation makes bug tracking in either of the two classes a little easier compared to essentially navigating two classes combined into one. The second reason was doing this separation gave me a detailed understanding of how exactly the Progress Panel worked and how it interacted with other objects. The understanding gained here made developing a new display system for the panel much easier.

Design of New Implementation

The ASSISTments tutor is written in the Java programming language. The design philosophy of multiple presenter classes adhering to object-oriented design concepts was followed. The structure starts with an abstract class called ProgressHeaderPresenter. This parent class contains the fields and methods that each child class will inherit. A header presenter class was then created for each of the problem set types the system would encounter. They are as follows:

1. HomoHeaderPresenter – Linear/random complete all problem sets
2. IndeterminedHeaderPresenter – Placements problem sets

3. SkillBuilderHeaderPresenter – Skill Builder problem sets
4. LinearMixedHeaderPresenter – Linear problem set with sections containing mixed types of headers
5. RandomMixedHeaderPresenter – Random problem set with sections containing mixed types of headers
6. SkillBuilderMixedHeaderPresenter – Problem set containing sections of Skill Builders
7. IfProblemSetHeaderPresenter – If-then-else problem set
8. PlacementsHeaderPresenter – Placements problem set
9. ChooseProblemSetHeaderPresenter – Problem set with a choose condition

The last class created was Section to represent the sections the Progress Panel would have to display. A section has a root problem that it is attached to and a list of all the problems contained within its limits.

The ProgressHeaderPresenter base class provides a few methods to its derived classes. The method chooseHeaderPresenter takes a Section as an argument and returns the corresponding header presenter based on the problem set types inside of that section. A Boolean field inside the base class is used to keep track of which problems in a Section have been completed and ultimately whether or not the Section has been completed. The checkSection method checks to see whether all Sections in a problem set are covered. This is used to help keep track of when to display the "More to come…" label.

Each Presenter subclass overrides a set of abstract functions from the base class and the implementation corresponds to the type of problem set. The setIntitialProgressPanelHeader method sets the initial display of the header and the bookkeeping method is used to update that header display as the student completes the problems in that Section. The finishAction method handles when the “More to come…” label can be removed. The last and most interesting method is setUpHeader. The more complex header classes such as the LinearMixedHeaderPresenter contain a HashMap field that maps Manifests (problems) to Sections. In those classes a helper function buildSections is used to create the Sections that will be displayed by the Progress Panel. The HashMap is updated as the Sections are created and then setUpHeader uses those mappings to create that various headers it needs.

The last important piece of information is the use of a stack to keep track of everything. Header Presenters are pushed onto the stack when they are created. The previously mentioned bookKeeping method inside of the LinearMixedHeaderPresenter and RandomMixedHeaderPresenter can then make a call to the stack function peek() to access the presenters as necessary to perform its display updating function. The importance of this is that in complicated problem sets with multiple types of sections, the use of peek() allows the bookkeeping to be delegated to the children.

Results

With the new infrastructure in place we can now view some examples of what the Progress Panel looks like as we progress through a problem set. The problem set that was detailed in the Progress Panel section of the paper will be used to show these changes. Figure 4 shows the initial state of the Progress Panel upon loading the problem set in the tutor. The top-most header displays the total number of problems completed and since this is at the start that number is zero. The lone problem at the start of the set is placed into its own Section and is shown by the header stating that 0/1 problems have been completed. Finally the “More to come…” label is displayed informing the student that there is more to do after the current Section is completed.

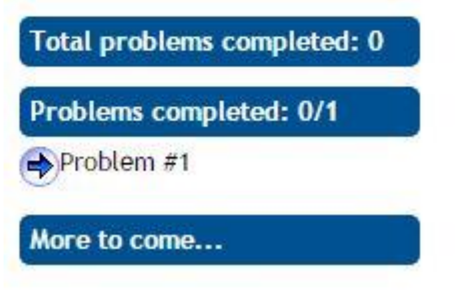


Figure 4: Starting Problem Set

Here the first Section has been completed and the student is now in the second section which is a skill builder. The top header has been updated with the number of problems completed so far, in this case one. The header for the second Section tells the student that they are in a Skill Builder and need to get three correct answers in a row to complete that Section. Finally the “More to come…” label is still present as there is more Sections after the current one.

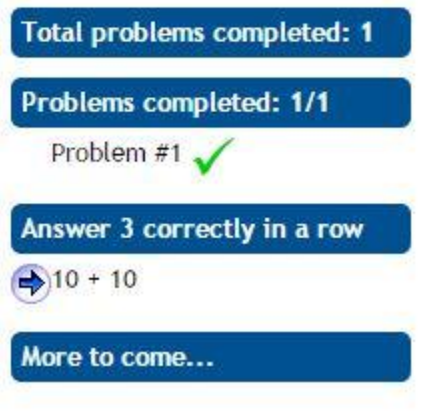


Figure 5: First Skill Builder Section

Here the student has entered the third Section which is made up of two single problems. Once again the top header has been updated to reflect the total number of problems completed. This Section's header therefore shows the student that there are two problems in this Section they need to complete.

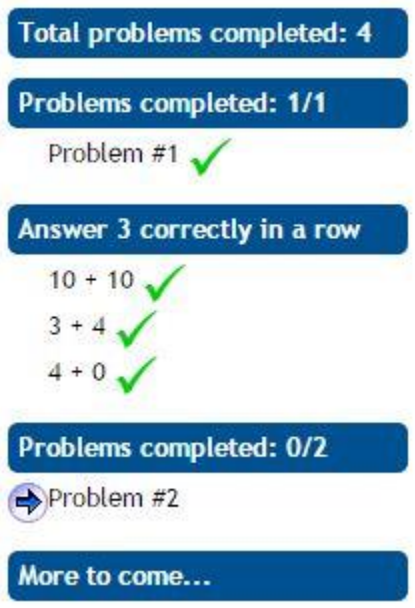
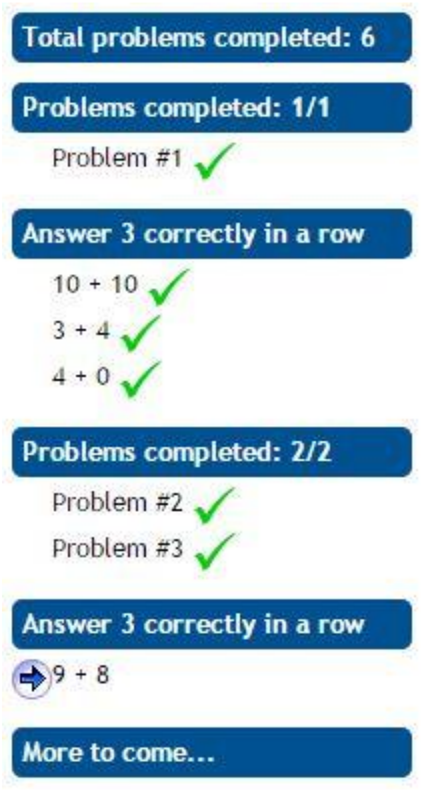


Figure 6: Section of Two Single Problems

The student has again entered a Skill Builder Section and the new header reflects that.


Total problems completed: 6

Problems completed: 1/1

Problem #1 ✓

Answer 3 correctly in a row

10 + 10 ✓

3 + 4 ✓

4 + 0 ✓

Problems completed: 2/2

Problem #2 ✓

Problem #3 ✓

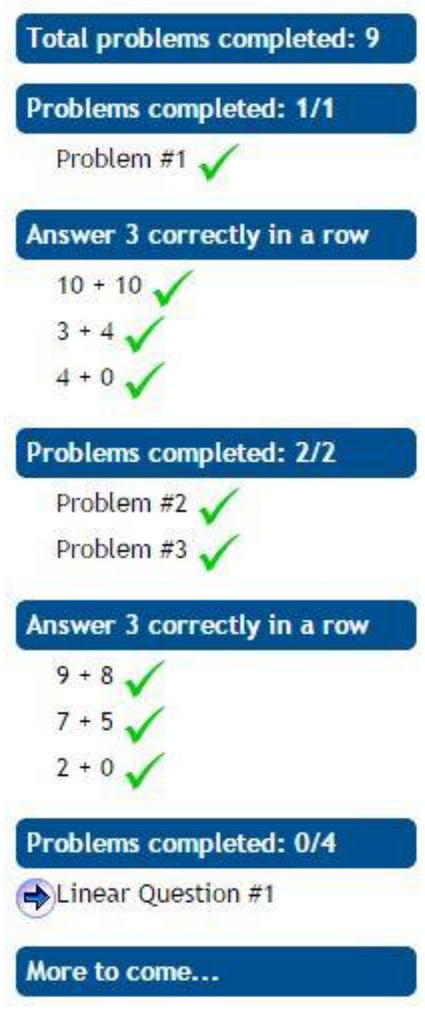
Answer 3 correctly in a row

9 + 8

More to come...

Figure 7: Second Skill Builder Section

Now the student is in a Linear Complete All Section. The header shows the student there is four total problems to complete before entering the next Section.



Total problems completed: 9

Problems completed: 1/1

Problem #1 ✓

Answer 3 correctly in a row

10 + 10 ✓

3 + 4 ✓

4 + 0 ✓

Problems completed: 2/2

Problem #2 ✓

Problem #3 ✓

Answer 3 correctly in a row

9 + 8 ✓

7 + 5 ✓

2 + 0 ✓

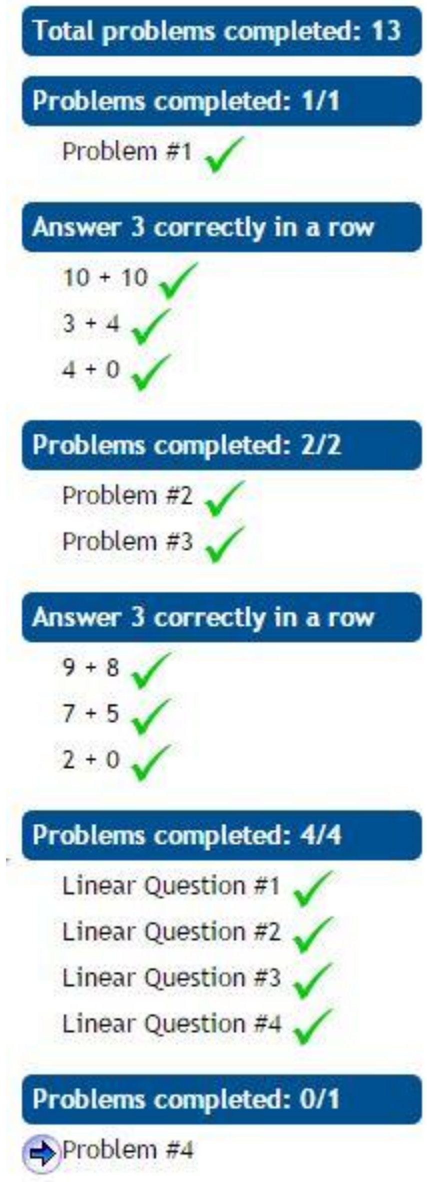
Problems completed: 0/4

Linear Question #1

More to come...

Figure 8: Linear Problem Set Section

Finally the student enters the last Section of the problem set. The “More to come…” label is no longer present informing them that once they complete the current Section they will be done with the entire problem set.


Total problems completed: 13

Problems completed: 1/1

Problem #1 ✓

Answer 3 correctly in a row

10 + 10 ✓

3 + 4 ✓

4 + 0 ✓

Problems completed: 2/2

Problem #2 ✓

Problem #3 ✓

Answer 3 correctly in a row

9 + 8 ✓

7 + 5 ✓

2 + 0 ✓

Problems completed: 4/4

Linear Question #1 ✓

Linear Question #2 ✓

Linear Question #3 ✓

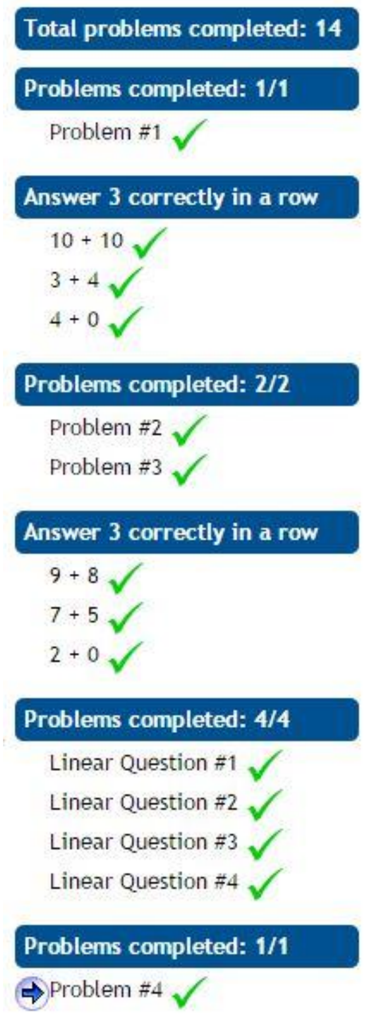
Linear Question #4 ✓

Problems completed: 0/1

Problem #4

Figure 9: Last Problem

Finally we have what the Progress Panel looks like when the entire problem set is complete. The top header shows the final number of problems the student completed. In this case where the student answered all the questions correctly that number is 14. That number is the minimum they could have done for this problem set. If that student had answered the first question in one of the Skill Builder Sections incorrectly and then subsequently answered the next three correctly, their total would be 15. Remember Skill Builders require a student to answer x questions correctly in a row, so different students could end up with different total counts upon completion.



Total problems completed: 14

Problems completed: 1/1

Problem #1 ✓

Answer 3 correctly in a row

10 + 10 ✓
3 + 4 ✓
4 + 0 ✓

Problems completed: 2/2

Problem #2 ✓
Problem #3 ✓

Answer 3 correctly in a row

9 + 8 ✓
7 + 5 ✓
2 + 0 ✓

Problems completed: 4/4

Linear Question #1 ✓
Linear Question #2 ✓
Linear Question #3 ✓
Linear Question #4 ✓

Problems completed: 1/1

Problem #4 ✓

Figure 10: Problem Set Completed

Future Work

The ASSISTments system is capable of handling the currently implemented types of problem sets. Over the years the number of problem types available in ASSISTments has grown and it is reasonable to believe that more may be added in the future. If this is the case, the structure and format I have created could be easily followed to add new presenters to match new types.

As far as improvements to the system currently, I can see a potential to add an indicator of a minimum number of problems needed to complete a problem set. Skill Builders in particular create a problem where there is no way to tell how many problems a student will have to complete because you have no idea if a student will get x questions correct in a row right at the start or if there will be incorrect answers mixed in there. It may be possible that each Section could be looked at, the Sections with concrete numbers of problems could return those numbers, and the Sections without could return the minimum number of problems to complete that Section. This could create a few new problems however. First is the potential to clutter the interface itself. Displaying this number to students would require another label to be displayed somewhere in the Progress Panel. Secondly and perhaps more importantly is it could possibly be disheartening for students who do not complete the problem set with ease. For example if the minimum number of problems to complete a problem set is 30, student A may see that, get every question correct on the first try, and be done at 30 questions completed. Now imagine student B does not do as well on the problem set. When they start they see that the minimum is 30 problems yet while doing it they reach 30 problems completed and there is still more to do. Student B may see this and get discouraged especially as their number of problems completed potentially continues to outgrow the minimum.

Conclusion

The updated Progress Panel should provide students with a better representation of where they are in a problem set and a better indication of completion. The updated implementation of the ProgressPanelPresenter may make debugging and future changes to the ApplicationPresenter and ProgressPanelPresenter easier without the two being intertwined. Additionally, any new problem types should be able to be integrated with relative ease following the structure created in this project.

Appendix

Source Code

org.assistments.gwt.student.tutor.presenter.ApplicationPresenter.java

```
/*****************************************************************************
***
 * Copyright (c) 2010, 2011 Worcester Polytechnic Institute
 * All rights reserved.

*****************************************************************************
**/
package org.assistments.gwt.student.tutor.presenter;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.assistments.core.domain.Action;
import org.assistments.core.domain.ProgressState;
import org.assistments.core.domain.PropertyKeys;
import org.assistments.core.domain.content.Manifest;
import org.assistments.core.domain.content.assess.Problem;
import org.assistments.core.domain.content.assess.SubmittableProblem;
import org.assistments.core.domain.content.assist.TutorStrategies;
import org.assistments.core.persistence.IsPersistable;
import org.assistments.core.persistence.Persistable;
import org.assistments.core.persistence.PersistenceManager;
import org.assistments.core.service.config.ConfigurationService;
import org.assistments.core.service.context.ApplicationContextService;
import org.assistments.core.service.context.NetworkState;
import org.assistments.core.service.recovery.RecoveryServiceNotifier;
import org.assistments.gwt.core.client.event.BaseEvent;
import org.assistments.gwt.core.client.event.GetPersistableEvent;
import org.assistments.gwt.core.client.event.GotPersistableEvent;
import org.assistments.gwt.core.client.event.GotPersistableEventHandler;
import org.assistments.gwt.core.client.event.RunOnLineInitEventFailure;
import
org.assistments.gwt.core.client.event.RunOnLineInitEventFailureHandler;
import org.assistments.gwt.core.client.event.action.UserActionEvent;
import org.assistments.gwt.core.client.event.action.UserActionEventHandler;
import
org.assistments.gwt.core.client.event.content.GetAssignmentEventFailure;
import
org.assistments.gwt.core.client.event.content.GetAssignmentEventFailureHandle
r;
import org.assistments.gwt.core.client.event.message.BaseErrorEvent;
import org.assistments.gwt.core.client.html5.HTML5FeatureTester.Feature;
import
org.assistments.gwt.core.client.html5.UnsupportedHTML5FeatureException;
import org.assistments.gwt.core.client.persistence.NoNetworkException;
```

```
import
org.assistments.gwt.core.client.persistence.impl.ClientAndServerPersistenceMa
nagerImpl;
import
org.assistments.gwt.core.client.persistence.impl.ServerPersistenceServiceImpl
;
import org.assistments.gwt.core.client.util.ClientUtil;
import org.assistments.gwt.core.util.Regex;
import org.assistments.gwt.core.view.DefaultMessage;
import org.assistments.gwt.student.tutor.domain.ProgressPanelResponseTypes;
import org.assistments.gwt.student.tutor.event.action.ActionType;
import org.assistments.gwt.student.tutor.event.action.LogStateException;
import
org.assistments.gwt.student.tutor.event.action.ExplanationSubmittedAction;
import org.assistments.gwt.student.tutor.event.action.ProblemFinishedAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemLimitExceededAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemSetExhaustedAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemSetFinishedAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemSetMasteredAction;
import org.assistments.gwt.student.tutor.event.action.StudentResponseAction;
import
org.assistments.gwt.student.tutor.event.action.StudentSubmissionAction;
import
org.assistments.gwt.student.tutor.event.action.TutoringSetStartedAction;
import
org.assistments.gwt.student.tutor.event.state.GetAssignmentStateEventFailure;
import
org.assistments.gwt.student.tutor.event.state.GetAssignmentStateEventFailureH
andler;
import
org.assistments.gwt.student.tutor.event.state.InitializeProgressPanelDataEven
tSuccess;
import
org.assistments.gwt.student.tutor.event.state.InitializeProgressPanelDataEven
tSuccessHandler;
import org.assistments.gwt.student.tutor.gin.NamedLocationProvider;
import
org.assistments.gwt.student.tutor.navigator.impl.ChooseConditionNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.IfThenElseNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.LinearNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.PlacementsNavigator;
import
org.assistments.gwt.student.tutor.presenter.devconsole.DeveloperConsolePresen
ter;
import
org.assistments.gwt.student.tutor.service.configuration.impl.StudentConfigura
tionServiceImpl;
import
org.assistments.gwt.student.tutor.service.context.StudentApplicationContextSe
rviceImpl;
import org.assistments.gwt.student.tutor.service.state.StateService;
import
org.assistments.gwt.student.tutor.terminator.impl.CorrectInARowTerminator;
```

```
import org.assistments.gwt.student.tutor.view.AffectDetectorView;
import org.assistments.gwt.student.tutor.view.ApplicationView;
import org.assistments.gwt.student.tutor.view.DownloadingDialog;
import org.assistments.gwt.student.tutor.view.FeaturesMissingDialog;
import
org.assistments.gwt.student.tutor.view.ProgressPanelInitializationData;
import org.assistments.gwt.student.tutor.view.WaitOrGoOfflineDialog;
import org.assistments.util.PriorityLevel;
import org.assistments.util.UIDGeneratorGWTImpl;
import org.assistments.util.Util;

import com.google.gwt.event.shared.EventHandler;
import com.google.gwt.event.shared.HandlerManager;
import com.google.gwt.storage.client.Storage;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.DisclosurePanel;
import com.google.gwt.user.client.ui.HasWidgets;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Image;
import com.google.gwt.user.client.ui.LayoutPanel;
import com.google.gwt.user.client.ui.Widget;
import com.google.inject.Inject;

public class ApplicationPresenter implements Presenter,
            GotPersistableEventHandler, GetAssignmentEventFailureHandler,
            GetAssignmentStateEventFailureHandler,
            RunOnLineInitEventFailureHandler,
            InitializeProgressPanelDataEventSuccessHandler,
            FeaturesMissingDialog.Presenter {

      private Manifest manifest;
      private String handlerUID;

      private ApplicationView view;
      private HandlerManager eventBus;
      private DownloadingDialog dialog;

  // Appears to be unused here, but we need it injected in order to properly
setup
  // (via injection) the various DevConsole presenters (look at
DeveloperConsolePresenter's constructor).
      private DeveloperConsolePresenter devConsolePresenter;

      private Map<String, ProgressPanelEntry> panelEntries;
      private ProgressPanelEntry mostRecentProgressPanelEntry;
      private Map<String, ProgressPanelEntry> pendingContentRequests;

      private ClientAndServerPersistenceManagerImpl pm;

      private StudentConfigurationServiceImpl configService;

      private WaitOrGoOfflineDialog waitOrGoOfflineDialog;

      private OfflineUserPresenter offlineUserPresenter;

      private NamedLocationProvider namedLocationProvider;
      private RecoveryServiceNotifier recoveryServiceNotifier;

      private AffectDetectorPresenter affectDetectorPresenter;
      private AffectDetectorView adview;

      //AS-38
      private StateService stateService;

      private boolean isSkillBuilder;
      private boolean isProblemCountIndeterminate;
      private StudentApplicationContextServiceImpl acs;

      // Presenter for Progress Panel
      private ProgressPanelPresenter ppp;

      // Header presenter used for this Problem Set
      boolean includingSection;
      private ProgressHeaderPresenter headerPresenter;


      @Inject
      public ApplicationPresenter(HandlerManager eventBus,
          ApplicationView view,
          DeveloperConsolePresenter devConsolePresenter,
            DownloadingDialog dialog,
            WaitOrGoOfflineDialog waitOrGoOfflineDialog,
            ClientAndServerPersistenceManagerImpl pm,
            ConfigurationService configService,
            OfflineUserPresenter offlineUserPresenter,
            RecoveryServiceNotifier recoveryServiceNotifier,
            AffectDetectorPresenter affectDetectorPresenter,
            ApplicationContextService acs){

            this.configService = (StudentConfigurationServiceImpl)
configService;
            this.offlineUserPresenter = offlineUserPresenter;
            this.recoveryServiceNotifier = recoveryServiceNotifier;
            this.handlerUID = UIDGeneratorGWTImpl.generateDefaultUID();
            this.eventBus = eventBus;
            this.view = view;
            this.view.setPresenter(this);
            this.devConsolePresenter = devConsolePresenter;
            this.dialog = dialog;
            this.waitOrGoOfflineDialog = waitOrGoOfflineDialog;
            this.affectDetectorPresenter = affectDetectorPresenter;
            this.adview = new AffectDetectorView();
            this.view.addAffectDetectorPanel(adview);
            this.affectDetectorPresenter.setView(adview);
            this.acs = (StudentApplicationContextServiceImpl) acs;

            waitOrGoOfflineDialog.setParent(this);
            panelEntries = new HashMap<String, ProgressPanelEntry>();
            pendingContentRequests = new HashMap<String,
ProgressPanelEntry>();
            this.pm = pm;
            ppp = new ProgressPanelPresenter(pm, view,
mostRecentProgressPanelEntry,
                        isSkillBuilder, isProblemCountIndeterminate, this,
eventBus);
            bind();
      }

      @Inject
      public void setStateService(StateService stateService){
            this.stateService = stateService;
      }

      @Inject
      public void setNamedLocationProvider(NamedLocationProvider
namedLocationProvider){
            this.namedLocationProvider = namedLocationProvider;
      }


      // userID, literally their user ID number, not currently used.
      public void setUserInfo(String userID, String displayName, String
loginName, List<String> roles){
            // this.userID = userID;
            view.setAccountName(displayName, loginName);
            view.setRoleAllowances(roles);
      }

  public void showAssignmentsLink()
  {
    // dm: AS-876
    // Show the "Assignments" link only if we are
        // - not in test drive, and
        // - not in public preview, and
        // - not being called by some 3rd party through the API, and finally,
        // - are working online.
        // Whew!
    if(!acs.isPreviewMode() && !acs.isPublicPreview() && !acs.hasOnExit() &&
acs.isWorkModeOnLine())
    {
      view.showAssignmentsLink();
    }
  }

      public void onAssignmentListLinkClicked()
      {

ClientUtil.exitApplication(namedLocationProvider.getNamedLocationForBackToAss
ignmentsList().getLocation());
      }

      private void bind() {
            eventBus.addHandler(InitializeProgressPanelDataEventSuccess.TYPE,
this);
            eventBus.addHandler(GotPersistableEvent.TYPE, this);
            eventBus.addHandler(GetAssignmentEventFailure.TYPE, this);
            eventBus.addHandler(GetAssignmentStateEventFailure.TYPE, this);
            eventBus.addHandler(RunOnLineInitEventFailure.TYPE, this);
```

```
	}

	// AS-951
	public void setManifest(Manifest manifest){
		this.manifest = manifest;
		includingSection = this.hasSection(manifest);
		this.headerPresenter = chooseHeaderPresenter(manifest);
	}

	// AS-951
	public ProgressHeaderPresenter chooseHeaderPresenter(Manifest 
manifest){
		//test if there is need to build our section hierarchy
		//requirement: Problem Set exists
		//if so, need to build the structure

		//If only a problem not a Problem Set
		if(manifest.isProblem()){
			return new HomoHeaderPresenter(manifest, 1, view);
		}
		//Placements will use IndeterminedHearderPresenter
		else if(this.checkManifestType(manifest).equals("Placements")){
			return new PlacementsHeaderPresenter(manifest, view);
		}
		//Choose type: we have a specific presenter for it
		else 
if(this.checkManifestType(manifest).equals("ChooseCondition")){
			return new ChooseProblemsetHeaderPresenter(manifest, view);
		}
		//If type: use RandomMixedHeaderPresenter temporarily, cause we 
don't know if there
	       //are any nested sections is randomized
		else if(this.checkManifestType(manifest).equals("IfThenElse")){
			return new IfProblemsetHeaderPresenter(manifest, view);
		}
		//if there are nested sections,  we should consider
		else if(this.hasSection(manifest)){
			Map<String,String> props;
		  String navigator;

		  props = manifest.getProperties();
		  navigator = props.get(PropertyKeys.NAVIGATOR_TYPE);
		  boolean isSkillBuilder = 
props.get(PropertyKeys.TERMINATOR_TYPE).equals(CorrectInARowTerminator.class.
getName());
		  if(isSkillBuilder){
		      return new SkillBuilderMixedHeaderPresenter(manifest, 
view);
		  }
		  // Random type: we use Random Presenter
		  else if(navigator.equals(LinearNavigator.NAVIGATOR_TYPE)){
		      return new LinearMixedHeaderPresenter(manifest, view);
		  }
		  // we assume others are all Linear type mixed presenters 
applied to
		  else{
		      return new RandomMixedHeaderPresenter(manifest, view);
```

```
              }
            }
            //No nested Sections, only one layer, use basic type presenter is
enough
            else{
              if(this.checkManifestType(manifest).equals("HomoProblemSet")){
                    return new HomoHeaderPresenter(manifest,
manifest.getDescendantProblemCount(),view);
              }
              else{
                    return new SkillBuilderHeaderPresenter(manifest,view);
              }
            }
        }

      // 951
      // Used to test if this tutor contains sections
      public boolean hasSection(Manifest manifest){
            boolean result = false;
            for(Manifest man : manifest.getChildManifests()){
                  if(man.isProblemSet()){
                        result = true;
                        break;
                  }
            }
            return result;
      }

      @Override
      public void go(HasWidgets container) {
            container.add(view);
      }

      /**
       * Upon initialization, add all the started problems, and all the
student
       * response actions.
       */
  @Override
  public void
handleInitializeProgressPanelDataEventSuccess(InitializeProgressPanelDataEven
tSuccess evt){
        // Assignment manifest is the root manifest.
        ProgressState assignmentState;

        try{
            String rootKey = (Manifest.getRoot(this.manifest)).getKey();
            assignmentState = stateService.getProgressStateFor(rootKey);

            switch(assignmentState){
              case COMPLETED:
              case EXHAUSTED:
                    case LIMIT_EXCEEDED:
                    case MASTERED:
                    case NOT_STARTED:
                          //Nothing to do yet with the Progress Panel.
                          break;
                    case IN_PROGRESS:
                          //Update the panel showing past work
                          ProgressPanelInitializationData data =
evt.getData();

                          for(Action action : data.getActions()){
                                ppp.updateProgressPanel(action);
                          }
                          break;
                    case UNDEFINED:
                          throw new IllegalStateException("Unexpected
UNDEFINED state in handleInitializeProgressPanelDataEventSuccess(): " +
rootKey);
                    default:
                          break;
            }
    }
    catch(LogStateException e){
      throw new IllegalStateException(e);
    }
  }

      /**
       * Issue an asynchronous content call to retrieve the body of the
problem.
       *
       * @param manKey
       */
      public void requestContent(String manKey) {
            ProgressPanelEntry entry = ppp.getPanelEntry(manKey);
            if (entry == null) {
                  return;
            }

            if (entry.isHasContent()) {
                  return;
            }

            Manifest man = (Manifest)
pm.retrieveOne(PersistenceManager.SL_CACHE,
                        manKey);
            pendingContentRequests.put(man.getContentKey(), entry);
            view.setPopupLoading(entry.getDisclosurePanel());
            eventBus.fireEvent(new GetPersistableEvent(man.getContentKey(),
                        getHandlerUID()));
      }

      @Override
      public void onGotPersistableEvent(GotPersistableEvent evt) {

            ProgressPanelEntry entry = pendingContentRequests.remove(evt
                        .getPersistenceKey());
            if (entry == null) {
                  return;
            }

            IsPersistable result = evt.getPersistable();
            if (result == null || !(result instanceof Problem)) {
                  return;
            }

            entry.setHasContent(true);
            Problem problem = (Problem) result;
            String questionHtml = problem.getQuestion();
            Manifest man = (Manifest)
pm.retrieveOne(PersistenceManager.SL_CACHE,
                        entry.getManifestKey());
            String assistmentID = man.getProperties().get(
                        PropertyKeys.ASSISTMENT_ID_ENCODED);

            String defaultTitle = view.getTitleString(questionHtml,
assistmentID);

            view.setPanelText(assistmentID, entry.getDisclosurePanel(),
                        questionHtml,
                        defaultTitle);
      }

      /**
       * Hide everything except for the developers console.
       */
      public void hideTutor() {
            view.clearAssignmentContainer();
      }

      /**
       * Retrieve the progress panel entry whose manifest has the given
tutoring
       * key.
       *
       * @param tutoringKey
       * @return
       */
      private ProgressPanelEntry getPanelEntryByTutoringKey(String
tutoringKey) {
            for (String manKey : panelEntries.keySet()) {
                  Manifest man = (Manifest) pm.retrieveOne(
                              PersistenceManager.SL_CACHE, manKey);
                  if (man != null && man.getTutoringKey() != null) {
                        // During replay, the tutor strategies may not be in
the cache yet
                        // have to check client storage as well
                        // Solution is to do this ugly blocking loop to wrap
the async call inside
                        // the synchronous one
                        PersistenceManagerAsyncCallWrapper wrapper = new
PersistenceManagerAsyncCallWrapper();
                        TutorStrategies ts = null;
                        try {
                              ts = (TutorStrategies)

      wrapper.retrieveOne(PersistenceManager.SL_CLIENT,
man.getTutoringKey());
                        }
                        catch (Throwable caught) {
                        }


                        if (ts.hasChildManifestKey(tutoringKey)) { // Does
the problem

            // have the given

            // tutoring root?
                              return panelEntries.get(manKey);
                        }
                  }
            }
            return null;
      }

      private class PersistenceManagerAsyncCallWrapper implements
AsyncCallback<IsPersistable> {

            private boolean busy = false;
            private Throwable caught;
            private IsPersistable result;

            public IsPersistable retrieveOne(int storageLevel, String
persistenceKey) throws Throwable {
                  this.busy = true;
                  pm.retrieveOne(storageLevel, persistenceKey, this);
                  while (busy) {
                        ; // Wait
                  }
                  if (caught != null) {
                        throw caught;
                  }
                  else {
                        return result;
                  }
            }

            @Override
            public void onFailure(Throwable caught) {
                  this.caught = caught;
                  this.busy = false;

            }

            @Override
            public void onSuccess(IsPersistable result) {
                  this.result = result;
                  this.busy = false;
            }
      }

      public LayoutPanel getAssignmentContainer() {
            return view.getAssignmentContainer();
      }
```

```
	public void showDownloadingDialog(String name) {
		dialog.setText("Downloading assignment \"" + name + "\".");
		dialog.center();
	}

	public void hideDownloadingDialog() {
		dialog.hide();
	}

	@Override
	public String getHandlerUID() {
		return handlerUID;
	}

	public void clearAssignmentContainer() {
		view.getAssignmentContainer().clear();
	}

	public void addToAssignmentPanel(Widget w) {
		view.addToAssignmentPanel(w);
	}

	@Override
	public void handleGetAssignmentEventFailure(GetAssignmentEventFailure
evt) {
		expectingNoNetworkException(evt);
	}

	@Override
	public void handleRunOnLineInitEventFailure(RunOnLineInitEventFailure
evt) {
		expectingNoNetworkException(evt);
	}

	@Override
	public void handleGetAssignmentStateEventFailure(
			GetAssignmentStateEventFailure evt) {
		expectingNoNetworkException(evt);
	}

	private void expectingNoNetworkException(BaseErrorEvent<?, ?> evt) {
		if (evt.getException() instanceof NoNetworkException) {
			showWaitOrGoOfflineDialog(evt);
		} else {
			// Don't know what it is, but can't be good.

	recoveryServiceNotifier.notifyRecoveryService(PriorityLevel.ERROR,
					evt.getException());
		}
	}

	public <H extends EventHandler, P extends EventHandler> void
showWaitOrGoOfflineDialog(
			BaseErrorEvent<H, P> failedEvent) {
		waitOrGoOfflineDialog.setFailedEvent(failedEvent);
		waitOrGoOfflineDialog.center();
	}
```

```
      public <H extends EventHandler, P extends EventHandler> void 
retryIfOnline(
                  final BaseErrorEvent<H, P> event) {
            ServerPersistenceServiceImpl
                        .isNetworkAlive(new AsyncCallback<NetworkState>() {

                              @Override
                              public void onFailure(Throwable caught) { // 
This should not

                  // be called
                              }

                              @Override
                              public void onSuccess(NetworkState result) {
                                    switch (result) {
                                    case CONNECTED:
                                          waitOrGoOfflineDialog.hide();
                                          BaseEvent<?> toRetry = 
event.getParentEvent();

                                          eventBus.fireEvent(toRetry);
                                          break;
                                    default:
                                          // Do nothing
                                    }
                              }
                        });
      }

      public void showOfflineUserDialog() {
            offlineUserPresenter.go(null);
      }

      public void indicateFeaturesMissing(UnsupportedHTML5FeatureException e) 
      {
            FeaturesMissingDialog dialog = new FeaturesMissingDialog(this,
                        true,
                        
      namedLocationProvider.getNamedLocationForBackToAssignmentsList().getNam
e());

            Map<Feature, Boolean> features = e.getFeatureMap();

            for (Map.Entry<Feature, Boolean> entry : features.entrySet()) {
                  dialog.addFeature(entry.getKey().getDisplayName(), 
entry.getValue());
            }

            dialog.center();
      }

      @Override
      public void onBackButtonClicked() {
            
      ClientUtil.exitApplication(namedLocationProvider.getNamedLocationForBac
kToAssignmentsList().getLocation());
```

```
      }

      public void showSettings() {
            // TODO eventually this should display a dialog with a variety
            // of options, but for now it simply lets you clear Web Storage
            boolean approved = Window.confirm("Clear local storage?");
            if (approved && Storage.isLocalStorageSupported()) {
                  Storage.getLocalStorageIfSupported().clear();
                  Window.Location.reload();
            }
      }

      public ProgressPanelPresenter getProgressPanelPresenter(){
            return this.ppp;
      }

      public Manifest getManifest(){
            return this.manifest;
      }

      public ProgressHeaderPresenter getProgressHeaderPresenter(){
            return this.headerPresenter;
      }

      // Helper method to determine the problem set
      String checkManifestType(Manifest man){
            String type;
            Map<String, String> props;
            String navigator;

            props = man.getProperties();
            navigator = props.get(PropertyKeys.NAVIGATOR_TYPE);
            if(man.isProblem()){
                  type = "Problem";
            }
            else
if(props.get(PropertyKeys.TERMINATOR_TYPE).equals(CorrectInARowTerminator.cla
ss.getName())){
                  type = "SkillBuilder";
            }
            else
if(navigator.equals(ChooseConditionNavigator.NAVIGATOR_TYPE)){
                type = "ChooseCondition";
            }
            else if(navigator.equals(IfThenElseNavigator.NAVIGATOR_TYPE)){
                type = "IfThenElse";
            }
            else if(navigator.equals(PlacementsNavigator.NAVIGATOR_TYPE)){
                type = "Placements";
            }
            else{
                type = "HomoProblemSet";
            }
            return type;
      }
}
```

org.assistments.gwt.student.tutor.presenter.ProgressPanelPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import java.util.HashMap;
import java.util.Map;

import org.assistments.core.domain.Action;
import org.assistments.core.domain.ProgressState;
import org.assistments.core.domain.PropertyKeys;
import org.assistments.core.domain.content.Manifest;
import org.assistments.core.domain.content.assess.Problem;
import org.assistments.core.domain.content.assess.SubmittableProblem;
import org.assistments.core.persistence.IsPersistable;
import org.assistments.core.persistence.Persistable;
import org.assistments.core.persistence.PersistenceManager;
import org.assistments.gwt.core.client.event.action.UserActionEvent;
import
org.assistments.gwt.core.client.persistence.impl.ClientAndServerPersistenceMa
nagerImpl;
import org.assistments.gwt.core.util.Regex;
import org.assistments.gwt.core.view.DefaultMessage;
import org.assistments.gwt.student.tutor.domain.ProgressPanelResponseTypes;
import org.assistments.gwt.student.tutor.event.action.ActionType;
import
org.assistments.gwt.student.tutor.event.action.ExplanationSubmittedAction;
import org.assistments.gwt.student.tutor.event.action.LogStateException;
import org.assistments.gwt.student.tutor.event.action.ProblemFinishedAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemLimitExceededAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemSetExhaustedAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemSetFinishedAction;
import
org.assistments.gwt.student.tutor.event.action.ProblemSetMasteredAction;
import org.assistments.gwt.student.tutor.event.action.StudentResponseAction;
import
org.assistments.gwt.student.tutor.event.action.StudentSubmissionAction;
import
org.assistments.gwt.student.tutor.event.action.TutoringSetStartedAction;
import
org.assistments.gwt.student.tutor.event.state.InitializeProgressPanelDataEven
tSuccess;
import
org.assistments.gwt.student.tutor.event.state.InitializeProgressPanelDataEven
tSuccessHandler;
import
org.assistments.gwt.student.tutor.navigator.impl.ChooseConditionNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.IfThenElseNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.LinearNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.PlacementsNavigator;
import org.assistments.gwt.student.tutor.service.state.StateService;
import
org.assistments.gwt.student.tutor.terminator.impl.CorrectInARowTerminator;
import org.assistments.gwt.student.tutor.view.ApplicationView;
import
org.assistments.gwt.student.tutor.view.ProgressPanelInitializationData;
import org.assistments.util.UIDGeneratorGWTImpl;
import org.assistments.util.Util;
import org.assistments.gwt.core.client.event.action.UserActionEvent;
import org.assistments.gwt.core.client.event.action.UserActionEventHandler;

import com.google.gwt.event.shared.HandlerManager;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.DisclosurePanel;
import com.google.gwt.user.client.ui.HasWidgets;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Image;
import com.google.inject.Inject;

public class ProgressPanelPresenter implements Presenter,
UserActionEventHandler{

      private ClientAndServerPersistenceManagerImpl pm;
      private ApplicationView view;
      private Map<String, ProgressPanelEntry> panelEntries;
      private ProgressPanelEntry mostRecentProgressPanelEntry;
      private HandlerManager eventBus;

      // Header presenter used for this Problem Set
      boolean includingSection;
      private ProgressHeaderPresenter headerPresenter;

      // The ApplicationPresenter that this ProgressPanelPresenter belongs to
      private ApplicationPresenter ap;

      @Inject
      public ProgressPanelPresenter(ClientAndServerPersistenceManagerImpl pm,
ApplicationView view,
                  ProgressPanelEntry mostRecentProgressPanelEntry, boolean
isSkillBuilder,
                  boolean isProblemCountIndeterminate, ApplicationPresenter
ap,
                  HandlerManager eventBus){
            this.pm = pm;
            this.view = view;
            this.mostRecentProgressPanelEntry = mostRecentProgressPanelEntry;
            this.ap = ap;
            this.panelEntries = new HashMap<String, ProgressPanelEntry>();
            this.eventBus = eventBus;
            bind();
      }

      @Override
      public void go(HasWidgets container) {
            container.add(view);
      }
```

```
	private void bind(){
		eventBus.addHandler(UserActionEvent.TYPE, this);
	}


	public void disableProgressPanel(){
		eventBus.removeHandler(UserActionEvent.TYPE, this);
		this.view.clearProgressPanel();
	}

	/**
	 * The progress panel is populated by listening for user actions.
	 */
	@Override
	public void onUserActionEvent(UserActionEvent evt)
	{
		Action action = evt.getAction();

		this.updateProgressPanel(action);
	}

	public void updateProgressPanel(Action action)
	{
	  ActionType aType = ActionType.getType(action);

	  if (action.isScaffolding())
      {
        return;
      }

      switch (aType)
      {
	case ProblemStartedAction:
	case ProblemResumedAction:
	  addProblemByManifestKey(action.getManifestKey(),
action.getTimestamp());
	  break;

	case StudentResponseAction:
	  addStudentResponseAction((StudentResponseAction) action);
	  break;

	case ExplanationSubmittedAction:
	  addExplanationSubmittedAction((ExplanationSubmittedAction) action);
	  break;
	case StudentSubmissionAction:
	  addStudentSubmissionAction((StudentSubmissionAction) action);
	  break;

	case TutoringSetStartedAction:
	  addTutoringRequestedAction((TutoringSetStartedAction) action);
	  break;

	case ProblemFinishedAction:
	  ProblemFinishedAction pfa = (ProblemFinishedAction) action;
```

```
        //CD 8/7/2015  Do not do anything if the entry is null.
        // Entry being null means that the problem was deleted and
        // we will throw a StructureChangeException once this
        // processing is done.
        if(mostRecentProgressPanelEntry != null)
        {
            progressPanelRecording(action);

            ProgressPanelIcon panelIcon =
ProgressPanelIcon.getIconStateFromName(pfa.getIcon());

            Image icon = panelIcon.getIconImage();

            HorizontalPanel panel = (HorizontalPanel)
mostRecentProgressPanelEntry.getDisclosurePanel().getParent();

            panel.add(icon);

        }

      default:
        break;
    }
  }

      /**
       * Add a problem to the progress panel by it's manifest key. If it
already
       * exists, update it's name.
       *
       * @param manKey
       * @param timestamp
       */
  public void addProblemByManifestKey(final String manKey, final long
timestamp){
        final Manifest man = (Manifest)
pm.retrieveOne(PersistenceManager.SL_CACHE, manKey);

        if(man != null){
              // Note:
            // In the key:
@org.assistments.core.domain.content.Manifest/PR537232-824087
            // 537232 is the assistment id
            // 824087 is the problem id
            // This is going to get the encoded version of the assistment id
(537232)
              final String encodedAssistmentId =
man.getProperties().get(PropertyKeys.ASSISTMENT_ID_DECODED);

              pm.retrieveOne(PersistenceManager.SL_CLIENT,
man.getContentKey(),
                          new AsyncCallback<IsPersistable>(){
                    @Override
                    public void onSuccess(IsPersistable result){
                          // AS-951
                          // Remove the label if exists
                          // will be added again if necessary
                          view.removeMoreToCome();
                          String problemQuestion = null;

                          // Default the real problem ID
                          String problemID =
UIDGeneratorGWTImpl.extractCoreUID(Persistable.extractUID(manKey));

                          // Friendlier version is the encoded ID users are
used to seeing
                          if(!Util.isNullOrEmpty(encodedAssistmentId)){
                                problemID = encodedAssistmentId;
                          }

                          if(result != null && result instanceof Problem){
                                // Trim and strip ensures that we don't end
up with nothing visible
                                // when the problem contains only images (and
now actual text).
                                problemQuestion = ((Problem)
result).getQuestion();
                          }

                          ProgressPanelEntry entry = getPanelEntry(manKey);

                          // If there isn't one registered yet, create a new
one.
                          if(entry == null){

                                // AS-951

ap.getProgressHeaderPresenter().setUpHeader(man);

                                DisclosurePanel panel = view.addProblem(
                                            manKey,
                                            problemID,
                                            problemQuestion,
                                            problemID);
                                entry = new ProgressPanelEntry(manKey);
                                entry.setDisclosurePanel(panel);
                                entry.setStartTime(timestamp);
                                if(result != null && result instanceof
Problem){

                                      entry.setHasContent(true);
                                }
                                panelEntries.put(manKey, entry);
                                mostRecentProgressPanelEntry = entry;
                          }
                          // dm TODO after the below TODO:
                          // Because at this time StudentResponseActions
coming from the server DO NOT
                          // contains the TranslationKeys.IS_FIRST_RESPONSE
field, it is likely this
                          // logic has to do with the recording the
correctness of the earliest response
                          // action - because it is by definition the first
response to the current
```

```
                              // problem. It would be more straight forward to
have the field.

                              // dm TODO: addPanelText is called within
view.addProblem() above.
                              // Can we just create the entry above (if one
doesn't exist), and
                              // always call setPanelText here? Why have the call
in two places?
                              // I actually suspect this is not so simple.
                              // Do we want the initial entry to always display
the encoded ID?
                              // To be replaced by the question snippet... only
after the mouse-over?
                              // It previously did that, but I'm managed to start
with the snippet
                              // when possible
                              else{
                                    view.setPanelText(
                                                problemID,
                                                entry.getDisclosurePanel(),
                                                problemQuestion,
                                                problemID);
                              }
                              markProblemActive(manKey);

                              if(ap.getProgressHeaderPresenter().moreToCome ==
true){
                                    view.showMoreToCome();
                              }
                        }

                        @Override
                        public void onFailure(Throwable caught){
                              throw new IllegalStateException(caught);
                        }
                  });
                  }
            }

  /**
   * Add a student response, specifying whether or not to show the
correctness
   * of the answer. This is used to account for test mode when resuming a
   * problem set.
   *
   * @param action
   */
  public void addStudentResponseAction(StudentResponseAction action)
  {
    ProgressPanelEntry entry = getPanelEntry(action.getManifestKey());

    if (entry == null)
    {
      return;
    }
    int deltaSec = getSecondsDelta(entry.getStartTime(),
action.getTimestamp());

    // If the answer is an image tag, we want to replace the incorrect
    // message
    String response = action.getResponse().toViewString();
    // Truncate all answers except hard-coded string for
    // img tag answers
    Boolean needToTrunc = true;
    if (Regex.containsImgTag(response) || response.isEmpty())
    {
      response = new DefaultMessage().toString();
      needToTrunc = false;
    }


    ProgressPanelResponseTypes pprt =
this.getProgressPanelResponseTypes(action);
    view.addAction(entry.getDisclosurePanel(), response, deltaSec, pprt,
needToTrunc);
    answerProblem(entry);

  }


  public void addExplanationSubmittedAction(ExplanationSubmittedAction
action)
      {
            ProgressPanelEntry entry =
getPanelEntry(action.getManifestKey());

            if (entry != null)
            {
                  //add image to progress panel

                  int deltaSec =
getSecondsDelta(entry.getStartTime(),action.getTimestamp());
                  //blank string for explanations for now, may change in
future
                  view.addAction(entry.getDisclosurePanel(), "", deltaSec,
ProgressPanelResponseTypes.EXPLANATION, true);
            }

      }

  /**
       * Adds a submission action for open response problems
       *
       * @param action
       */
      public void addStudentSubmissionAction(StudentSubmissionAction action)
{
            ProgressPanelEntry entry =
getPanelEntry(action.getManifestKey());
            if (entry == null) {
                  return;
            }
            int deltaSec = getSecondsDelta(entry.getStartTime(),
action.getTimestamp());

            view.addAction(entry.getDisclosurePanel(),
action.getStudentResponse(), deltaSec,
ProgressPanelResponseTypes.UNKNOWN_ANSWER, true);
            answerProblem(entry);
      }

      /**
       * Add a tutoring set started action to the progress panel.
       *
       * @param action
       */
      public void addTutoringRequestedAction(TutoringSetStartedAction action)
{

            String manifestKey = action.getManifestKey();
            String uid = Persistable.extractUID(manifestKey);

                                    // or

                                    // scaffolding
                  ProgressPanelEntry entry = mostRecentProgressPanelEntry;//
getPanelEntryByTutoringKey(manifestKey);
                  if (entry == null) {
                        return;
                  }

                  // Get the problem id from the last section of the manifest
key
                  String entryProblemId =
Persistable.extractUID(entry.getManifestKey()).split("-")[1];
                  if (!entryProblemId.equals(uid.substring(2))) {
                        return;
                  }

                  int deltaSec = getSecondsDelta(entry.getStartTime(),
                              action.getTimestamp());

                  //Get the Problem Manifest
                  final Manifest man = (Manifest) pm.retrieveOne(
                              PersistenceManager.SL_CACHE,
entry.getManifestKey());

                  if (man != null &&
SubmittableProblem.isSubmittable(man.getContentType()))
                  {
                        // If submittable problem, Make sure view shows blue
dot
                        view.addAction(entry.getDisclosurePanel(), "Tutoring
Requested", deltaSec, ProgressPanelResponseTypes.UNKNOWN_ANSWER, false);
                        answerProblem(entry);
                  }
                  else
                  {
                        view.addAction(entry.getDisclosurePanel(), "Tutoring
Requested", deltaSec, ProgressPanelResponseTypes.INCORRECT_ANSWER, false);
                        answerProblem(entry);
                  }
      }

      private ProgressPanelResponseTypes
getProgressPanelResponseTypes(StudentResponseAction action){

            ProgressPanelResponseTypes progressPanelResponseType =
ProgressPanelResponseTypes.UNKNOWN_ANSWER;
            if (action.isShowCorrectness()){
                  if(action.isCorrect()){
                        progressPanelResponseType =
ProgressPanelResponseTypes.CORRECT_ANSWER;
                  }
                  else{
                        progressPanelResponseType =
ProgressPanelResponseTypes.INCORRECT_ANSWER;
                  }
            }

            return progressPanelResponseType;
      }

      /**
       * Mark a question correct.
       *
       * @param entry
       * @param correct
       */
      private void answerProblem(ProgressPanelEntry entry)
      {
            if (!entry.isAnswered())
            {
                  entry.setAnswered(true);
            }
      }

      private int getSecondsDelta(long first, long second) {
            long delta = second - first;
            int deltaSeconds = (int) delta / 1000;
            return deltaSeconds;
      }

  /**
       * Retrieve the disclosure panel associated with the given problem
manifest.
       *
       * @param manifest
       * @return
       */
      public ProgressPanelEntry getPanelEntry(String manifestKey) {
            ProgressPanelEntry panel = panelEntries.get(manifestKey);
            if (panel == null) {
                  for (String manKey : panelEntries.keySet()) {
                        if (manKey.equals(manifestKey)) {
                              // This means the manifest object wasn't in the
map, but a
                              // manifest
                              // with the same key WAS in the map. This is a
problem.
                              throw new IllegalStateException(
                                          "We have two instances of Manifest:
" + manifestKey);
                        }
                  }
            }
            return panel;
      }

      /**
       * Set a problem to "active" which puts a blue arrow next to the
heading of
       * the problem's disclosure panel.
       *
       * @param manifestKey
       */
      public void markProblemActive(String manifestKey) {
        // dm TODO: Seems inefficient.
        // Rather than turning everything off and then setting the current
one on,
        // How about we keep track of the previous entry and the current.
        // Turn off the previous, turn on the current.
            for (String mKey : panelEntries.keySet()) {

      view.markItemActive(getPanelEntry(mKey).getDisclosurePanel(), false);
            }

      view.markItemActive(getPanelEntry(manifestKey).getDisclosurePanel(),
                        true);
      }

      // 951
      /* This is a finish action for the problem set
     * For those simple problem set composed of only problems
     * no finish action
     * For those problem set which has "Total problems completed header
     * need to increment the top header counter because it starts from -1
     *  */
      private void progressPanelRecording(Action action){
            String manstr = action.getManifestKey();
            Manifest man = (Manifest)
pm.retrieveOne(PersistenceManager.SL_CACHE, manstr);

            if(man == null){
                  return;
            }
            if(action instanceof ProblemFinishedAction){
                  ap.getProgressHeaderPresenter().bookKeeping(man);
            }
            else if(action instanceof ProblemSetFinishedAction){
                  // for problem set which has "Total problems completed:..."
header
```

```
                    // increment the counter, because we start from 0 rather
than 1
                    if(ap.getProgressHeaderPresenter().manifest.equals(man)){
                        ap.getProgressHeaderPresenter().finishAction();
                    }
                }
                else{
                    if( (action instanceof ProblemSetExhaustedAction) ||
                        (action instanceof ProblemSetMasteredAction) ||
                        (action instanceof ProblemLimitExceededAction)){
                        String manType = ap.checkManifestType(man);
                        if(ap.includingSection &&
                                (manType.equals("SkillBuilder") ||
manType.equals("SkillBuilderMixed"))){

        if(!ap.getProgressHeaderPresenter().trackProgress.empty()){
                                ProgressHeaderPresenter php =
ap.getProgressHeaderPresenter().trackProgress.peek();
                                php.moreToCome = false;

        if(ap.getProgressHeaderPresenter().moreToCome == false){
                                    this.view.removeMoreToCome();
                                }
                            }
                        }
                    }
                }
        }

}
```

org.assistments.gwt.student.tutor.presenter.ProgressHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import java.util.Stack;
import java.util.Map;

import org.assistments.core.domain.PropertyKeys;
import org.assistments.core.domain.content.Manifest;
import
org.assistments.gwt.student.tutor.navigator.impl.ChooseConditionNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.IfThenElseNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.LinearNavigator;
import org.assistments.gwt.student.tutor.navigator.impl.PlacementsNavigator;
import
org.assistments.gwt.student.tutor.terminator.impl.CorrectInARowTerminator;
import org.assistments.gwt.student.tutor.view.ApplicationView;

public abstract class ProgressHeaderPresenter{

    // Update the headers in view
    ApplicationView view;
    // Root problem set manifest
    Manifest manifest;
```

```
	// Signal for showing "MoreToCome" label, if true, show
	boolean moreToCome = false;
	// Common function for the top header
	abstract void setInitialProgressPanelHeader();
	// Common function for the updating of progress
	abstract void bookKeeping(Manifest man);
	// Boolean array to record covering information, if a child is covered
corresponding index
	// will be set to true
	boolean sectionfinished[];
	// sectionfinished helper used in first time checkSection
	boolean sectionSetUp = false;
	// Stack storing the headers used in this problem set
	// Simple case: Homo/Indetermined/skillBuilder - push itself into its
own stack
	Stack<ProgressHeaderPresenter> trackProgress = new
Stack<ProgressHeaderPresenter>();

	// Finish Action: For "MoreToCome" label
	abstract public void finishAction();
	// Finish Action: for "MoreToCome" label
	abstract public void setUpHeader(Manifest man);

	// Return the type of this problem set based on the manifest
	String checkManifestType(Manifest man){
		String type;
		Map<String, String> props;
		String navigator;
		String terminator;

		props = man.getProperties();
		navigator = props.get(PropertyKeys.NAVIGATOR_TYPE);
		boolean hasSection = this.view.getPresenter().hasSection(man);
		if(hasSection){
			boolean isSkillBuilder =
props.get(PropertyKeys.TERMINATOR_TYPE).equals(CorrectInARowTerminator.class.
getName());
			if(isSkillBuilder){
				type = "SkillBuilderMixed";
			}
			else if(navigator.equals(LinearNavigator.NAVIGATOR_TYPE)){
				type = "LinearMixed";
			}
			else{
				type = "RandomMixed";
			}
		}
		else if(man.isProblem()){
			type = "Problem";
		}
		else
if(props.get(PropertyKeys.TERMINATOR_TYPE).equals(CorrectInARowTerminator.cla
ss.getName())){
			type = "SkillBuilder";
		}
		else
if(navigator.equals(ChooseConditionNavigator.NAVIGATOR_TYPE)){
```

```
                type = "ChooseCondition";
            }
            else if(navigator.equals(IfThenElseNavigator.NAVIGATOR_TYPE)){
                type = "IfThenElse";
            }
            else if(navigator.equals(PlacementsNavigator.NAVIGATOR_TYPE)){
                type = "Placements";
            }
            else{
                type = "HomoProblemSet";
            }
        return type;
        }

        // Check if all sections are covered, used in "MoreToCome" label
display
        void checkSection(Manifest m){
            // in first time, need to initialize the array
            if(sectionSetUp == false){
                sectionfinished = new
boolean[this.manifest.getChildCount()];
                for(int i = 0; i < sectionfinished.length; i++){
                    sectionfinished[i] = false;
                }
                sectionSetUp = true;
            }
            Manifest child;
            // Check which section a problem belongs to when it is being
answered
            // and color that section to "True"
            for(int i = 0; i < this.manifest.getChildCount(); i++){
                child = this.manifest.getChildManifests().get(i);
                // if this section has been colored, no need to set again
                if(sectionfinished[i] == true){
                    continue;
                }
                // this section is a problem itself
                if(child.isProblem() && child.equals(m)){
                    sectionfinished[i] = true;
                    break;
                }
                // this section is made of a lot of problems
                if(child.getAllDescendantProblemManifests().contains(m)){
                    sectionfinished[i] = true;
                    break;
                }
            }
            int checksize = 0;

            // count the covered sections
            for(int j = 0; j < sectionfinished.length; j++){
                if(sectionfinished[j] == true){
                    checksize++;
                }
            }
            // All sections are covered, heading to the last section of
problemset
```

```
            // No need to show "MoreToCome" label
            if(checksize == sectionfinished.length){
                  moreToCome = false;
            }
      }

      public ProgressHeaderPresenter chooseHeaderPresenter(Section s) {
            // test if there is a need to build our section hierarchy
            // requirement: problem set exists
            // if so need to build the structure
            boolean ifProblemset = s.type.equals("IfThenElse");
            boolean ChooseProblemset = s.type.equals("ChooseCondition");
            boolean Placements = s.type.equals("Placements");
            boolean LinearMixed = s.type.equals("LinearMixed");
            boolean RandomMixed = s.type.equals("RandomMixed");
            boolean SkillBuilderMixed = s.type.equals("SkillBuilderMixed");
            boolean isSkillBuilder = s.type.equals("SkillBuilder");
            boolean isHomo = s.type.equals("HomoProblemSet");

            // priority: Placements, Choose, If > others
            if(s.type.equals("Problem")){
                  return new HomoHeaderPresenter(s.belongsTo, 1, view);
            }
            else if(Placements){
                  return new PlacementsHeaderPresenter(s.belongsTo, view);
            }
            // Choose type: we have a specific presenter for it
            else if(ChooseProblemset){
                  return new ChooseProblemsetHeaderPresenter(s.belongsTo,
view);
            }
            // If type: use RandomMixedHeaderPresenter temporarily
            // we don't know if there are nested sections
            else if(ifProblemset){
                  return new IfProblemsetHeaderPresenter(s.belongsTo, view);
            }
            else if(isSkillBuilder){
                  return new SkillBuilderHeaderPresenter(s.belongsTo, view);
            }
            else if(isHomo){
                  //return new HomoHeaderPresenter(s.belongsTo,
s.children.size(), view);
                  return new HomoHeaderPresenter(s.belongsTo,
s.calcProblemTotal(), view);
            }
            // if there are nested sections we should consider
            else if(LinearMixed){
                  return new LinearMixedHeaderPresenter(s.belongsTo, view);
            }
            // No nested Sections, only one layer, basic presenter is enough
            else if(RandomMixed){
                  return new RandomMixedHeaderPresenter(s.belongsTo, view);
            }
            else if(SkillBuilderMixed){
                  return new SkillBuilderMixedHeaderPresenter(s.belongsTo,
view);
            }
            // No applied rules? do nothing...
            return null;
      }
}
```

org.assistments.gwt.student.tutor.presenter.Section.java

```
package org.assistments.gwt.student.tutor.presenter;

import java.util.ArrayList;

import org.assistments.core.domain.content.Manifest;

public class Section{
      // This section belongs under this manifest
      Manifest belongsTo;
      // Type of this section
      String type;
      // Elements in this section
      ArrayList<Manifest> children;
      // Number of Problems
      private int problemCount;

      public Section(String t){
            type = t;
            children = new ArrayList<Manifest>();
      }

      public void addChild(Manifest m){
            children.add(m);
      }

      public void removeChild(Manifest m){
            children.remove(m);
      }

      public void setProblemCount(int count){
            this.problemCount = count;
      }

      public int calcProblemTotal(){
            return children.size() - problemCount;
      }

}
```

org.assistments.gwt.student.tutor.presenter.HomoHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

// Linear/Random Problem set
public class HomoHeaderPresenter extends ProgressHeaderPresenter{

      // used in the top header
      private int totalProblems;
```

```
	private int currentRead = 0;

	public HomoHeaderPresenter(Manifest man, int total, ApplicationView v){
		manifest = man;
		this.view = v;
		totalProblems = total;
		moreToCome = false;
		String thisID = manifest.getUID();
		view.addSectionView(thisID);
		setInitialProgressPanelHeader();
		this.trackProgress.push(this);
	}

	@Override
	void setInitialProgressPanelHeader(){
		String display = "Problems completed: " + currentRead + "/" +
totalProblems;
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	void bookKeeping(Manifest man){
		currentRead++;
		String display = "Problems completed: " + currentRead + "/" +
totalProblems;
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	public void finishAction(){
		// TODO Auto-generated method stub

	}

	@Override
	public void setUpHeader(Manifest man){
		// TODO Auto-generated method stub

	}
}
```

org.assistments.gwt.student.tutor.presenter.SkillBuilderHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import org.assistments.core.domain.PropertyKeys;
import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

// Class for skill builder / answer x correctly in a row header
public class SkillBuilderHeaderPresenter extends ProgressHeaderPresenter{

	int masterProblems;

	public SkillBuilderHeaderPresenter(Manifest m, ApplicationView v){
		manifest = m;
```

```
            view = v;
            masterProblems = 
Integer.parseInt(manifest.getProperties().get(PropertyKeys.TERMINATOR_MASTERY
LIMIT));
            moreToCome = true;
            String thisID = manifest.getUID();
            view.addSectionView(thisID);
            setInitialProgressPanelHeader();
            this.trackProgress.push(this);
      }

      @Override
      void setInitialProgressPanelHeader(){
            String display = "Answer " + masterProblems + " correctly in a 
row";
            view.findSetSectionViewByID(manifest.getUID(), display);
            // if it's a simple skill builder, we need to switch off the 
moreToCome
            // Or, we should consider it to be a "uncertain" section, similar 
to LinearMixed
            // and RandomMixed
            if(this.manifest.equals(this.view.getPresenter().getManifest())){
                  this.moreToCome = false;
            }
      }

      @Override
      void bookKeeping(Manifest man){
            // We do nothing because the header will never be changed
            
      }

      @Override
      public void finishAction(){
            this.moreToCome = false;
      }

      @Override
      public void setUpHeader(Manifest man){
            // TODO Auto-generated method stub
            
      }

}
```

org.assistments.gwt.student.tutor.presenter.PlacementsHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

public class PlacementsHeaderPresenter extends ProgressHeaderPresenter {
      private int currentRead = 0;
      //HashMap<Manifest,Section> MappingSections;
```

```
	public PlacementsHeaderPresenter(Manifest man, ApplicationView v){
		manifest = man;
		view = v;
		moreToCome = false;
		String thisID = manifest.getUID();
		view.addSectionView(thisID);
		setInitialProgressPanelHeader();
	}

	@Override
	void setInitialProgressPanelHeader(){
		String display = "Problems completed: " + currentRead;
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	void bookKeeping(Manifest man){
		currentRead++;
		String display = "Problems completed: " + currentRead;
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	public void finishAction(){
		// TODO Auto-generated method stub

	}

	@Override
	public void setUpHeader(Manifest man){
		// TODO Auto-generated method stub

	}

}
```

org.assistments.gwt.student.tutor.presenter.IndeterminedHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

public class IndeterminedHeaderPresenter extends ProgressHeaderPresenter{

	// record the "x" filed in this header
	private int currentRead = 0;

	public IndeterminedHeaderPresenter(Manifest man, ApplicationView v){
		manifest = man;
		view = v;
		moreToCome = false;
		String thisID = manifest.getUID();
		view.addSectionView(thisID);
		setInitialProgressPanelHeader();
		this.trackProgress.push(this);
	}

	@Override
	void setInitialProgressPanelHeader(){
		String display = "Problems completed: " + currentRead;
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	void bookKeeping(Manifest man){
		currentRead++;
		String display = "Problems completed: " + currentRead;
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	public void finishAction(){
		// TODO Auto-generated method stub

	}

	@Override
	public void setUpHeader(Manifest man){
		// TODO Auto-generated method stub

	}

}
```

org.assistments.gwt.student.tutor.presenter.IfProblemsetHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import java.util.HashMap;

import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

/***
 * This is a If problemSet :
 * only three branches: IF/ Then/ Else
 * we have two headers pattern, one for the If part,
 * another for the then/else part
 * According to which branch is stepped in,choose that presenter to create
 */
public class IfProblemsetHeaderPresenter extends ProgressHeaderPresenter{

	ProgressHeaderPresenter headerPatternUsed;
	ProgressHeaderPresenter iFPartheaderPatternUsed;
	ProgressHeaderPresenter BranchPartheaderPatternUsed;
	HashMap<Manifest,Section> MappingSections;
	boolean Criteria = true;
	int currentRead = 0;

	public IfProblemsetHeaderPresenter(Manifest m, ApplicationView v){
		this.manifest = m;
```

```
        this.view = v;
        /**Currently, IF problemset Moretocome switch is true*/
        this.moreToCome = true;
        MappingSections = new HashMap<Manifest,Section>();
        for(Manifest m1 : this.manifest.getChildManifests()){
              buildTree(m1);
        }
        if(this.manifest.equals(view.getPresenter().getManifest())){
              String thisID = manifest.getUID();
              view.addSectionView(thisID);
              setInitialProgressPanelHeader();
        }
        iFPartheaderPatternUsed = this.view.getPresenter().

chooseHeaderPresenter(this.manifest.getChildManifests().get(0));
        headerPatternUsed = iFPartheaderPatternUsed;

        this.trackProgress=this.headerPatternUsed.trackProgress;
}

void buildTree(Manifest man){
        if(man.isProblem()){
              Section sec = new Section("Problem");
              sec.belongsTo = man;
              MappingSections.put(man,sec);
              sec.addChild(man);
        }
        else{
              String type = this.checkManifestType(man);
              Section sec = new Section(type);
              sec.belongsTo = man;
              for(Manifest m : man.getAllDescendantProblemManifests()){
                    MappingSections.put(m,sec);
                    sec.addChild(m);
              }
        }
}

@Override
void setInitialProgressPanelHeader(){
        String display = "Total problems completed: 0";
        view.findSetSectionViewByID(manifest.getUID(), display);
}

@Override
void bookKeeping(Manifest man){
        currentRead++;
        String display = "Total problems completed: " + currentRead;
        view.findSetSectionViewByID(manifest.getUID(), display);

        headerPatternUsed.bookKeeping(man);
        if(Criteria == false)
        moreToCome = this.headerPatternUsed.moreToCome;
}

@Override
public void finishAction(){
			this.moreToCome = false;
			if(this.manifest.equals(view.getPresenter().getManifest())){
				view.removeMoreToCome();
			}
		}

		@Override
		public void setUpHeader(Manifest man){
			if(Criteria == true){
				Section sec = this.MappingSections.get(man);

		if(!sec.belongsTo.equals(this.manifest.getChildManifests().get(0))){
					Criteria = false;
					BranchPartheaderPatternUsed =
this.view.getPresenter().chooseHeaderPresenter(sec.belongsTo);
					headerPatternUsed = this.BranchPartheaderPatternUsed;
					this.trackProgress =
this.BranchPartheaderPatternUsed.trackProgress;
				}
			}
			this.trackProgress = this.headerPatternUsed.trackProgress;
			boolean headerPatternUsedIsMixed = headerPatternUsed instanceof
LinearMixedHeaderPresenter
					|| headerPatternUsed instanceof
RandomMixedHeaderPresenter
					|| headerPatternUsed instanceof
SkillBuilderMixedHeaderPresenter
					|| headerPatternUsed instanceof
IfProblemsetHeaderPresenter
					|| headerPatternUsed instanceof
ChooseProblemsetHeaderPresenter;
			if(headerPatternUsedIsMixed){
				headerPatternUsed.setUpHeader(man);
			}
			if(Criteria == true)
				this.moreToCome = this.moreToCome ||
this.headerPatternUsed.moreToCome;
			else
				this.moreToCome = this.headerPatternUsed.moreToCome;
		}
}
```

org.assistments.gwt.student.tutor.presenter.ChooseProblemSetHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import java.util.HashMap;

import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

public class ChooseProblemsetHeaderPresenter extends ProgressHeaderPresenter{

	int currentRead = 0;
	ProgressHeaderPresenter headerPatternUsed;
	HashMap<Manifest, Section> MappingSections;
	boolean firstTimeHelper = false;
	boolean headerPatternUsedIsMixed = false;

	public ChooseProblemsetHeaderPresenter(Manifest m, ApplicationView v){
		this.manifest = m;
		view = v;
		moreToCome = false;
		MappingSections = new HashMap<Manifest,Section>();
		for(Manifest m1: this.manifest.getChildManifests()){
			buildTree(m1);
		}
	}

	void buildTree(Manifest man){
		if(man.isProblem()){
			Section sec = new Section("Problem");
			sec.belongsTo = man;
			MappingSections.put(man, sec);
			sec.addChild(man);
		}
		else{
			String type = this.checkManifestType(man);
			Section sec = new Section(type);
			sec.belongsTo = man;
			for(Manifest m : man.getAllDescendantProblemManifests()){
				MappingSections.put(m, sec);
				sec.addChild(m);
			}
		}
	}

	@Override
	void setInitialProgressPanelHeader(){
		// We dont know which section we will finish so put off
		// this step in bookkeeping
		String display = "Total problems completed: 0";
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	void bookKeeping(Manifest man){
		currentRead++;
		String display = "Total problems completed: " + currentRead;
		view.findSetSectionViewByID(manifest.getUID(), display);
		this.headerPatternUsed.bookKeeping(man);
		moreToCome = this.headerPatternUsed.moreToCome;
	}

	@Override
	public void finishAction(){
		this.moreToCome = false;
		if(this.manifest.equals(view.getPresenter().getManifest()) ==
true){
			this.view.removeMoreToCome();
		}

	}
	@Override
	public void setUpHeader(Manifest man){
		if(firstTimeHelper == false){
			Section sec = MappingSections.get(man);
			this.headerPatternUsed =
this.view.getPresenter().chooseHeaderPresenter(sec.belongsTo);
			this.trackProgress = this.headerPatternUsed.trackProgress;
			 headerPatternUsedIsMixed = headerPatternUsed instanceof
LinearMixedHeaderPresenter
					     || headerPatternUsed instanceof
RandomMixedHeaderPresenter
					     || headerPatternUsed instanceof
SkillBuilderMixedHeaderPresenter
					     || headerPatternUsed instanceof
IfProblemsetHeaderPresenter
					     || headerPatternUsed instanceof
ChooseProblemsetHeaderPresenter;

if(this.manifest.equals(view.getPresenter().getManifest())&&
headerPatternUsedIsMixed){
				String thisID = manifest.getUID();
				view.addSectionView(thisID);
				setInitialProgressPanelHeader();
			}
			firstTimeHelper=true;
		}
			this.headerPatternUsed.setUpHeader(man);
			this.moreToCome=this.headerPatternUsed.moreToCome;
	}
}
```

org.assistments.gwt.student.tutor.presenter.LinearMixedHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;


// Linear problem set containing sections composed of
// mixed headers
public class LinearMixedHeaderPresenter extends ProgressHeaderPresenter{

	// used in top header
	private int currentRead = 0;
	private int totalProblems;
	// problem -> Section
	HashMap<Manifest, Section> MappingSections;

	public LinearMixedHeaderPresenter(Manifest m, ApplicationView v){
		manifest = m;
```

```
            view = v;
            // MoreToCome label should be set to "true" except heading to
last section
            moreToCome = true;
            totalProblems = 0;
            MappingSections = new HashMap<Manifest,Section>();
            buildSections(manifest);
            String thisID = manifest.getUID();
            view.addSectionView(thisID);
            if(this.manifest.equals(view.getPresenter().getManifest())){
                  setInitialProgressPanelHeader();
            }
      }

      // for linear mixed problem set since we need to wrap those dangling
problems
      // to a new section, need to break the Parent-Child relations we did in
      // Build Tree Step and restructure it
      void buildSections(Manifest m){
            List<Manifest> children = m.getChildManifests();
            int problemCount = 0;
            for(int i = 0; i < children.size();){
                  // if problem set, check type, build corresponding section
                  if(children.get(i).isProblemSet()){
                        // Ignore on first Problem Set
                        if(i != 0){
                              // Check if current problem set and previous
problem set had multiple main problems
                              // If they did add current set to previous
section instead of a new section

      if((children.get(i).getProperties().get("multiPartProblem").equals("tru
e")) &&
                                    (children.get(i -
1).getProperties().get("multiPartProblem").equals("true"))){
                                    Section previous =
MappingSections.get(children.get(i -
1).getAllDescendantProblemManifests().get(0));
                                    for(Manifest man :
children.get(i).getAllDescendantProblemManifests()){
                                          MappingSections.put(man, previous);
                                          previous.addChild(man);

      previous.setProblemCount(problemCount++);
                                    }
                              }
                              else{
                                    String problemsetType =
this.checkManifestType(children.get(i));
                                    Section s = new Section(problemsetType);
                                    for(Manifest m1 :
children.get(i).getAllDescendantProblemManifests()){
                                          s.belongsTo = children.get(i);
                                          MappingSections.put(m1, s);
                                          s.addChild(m1);
                                    }
                              }
                        }

                        else{
                              String problemsetType =
this.checkManifestType(children.get(i));
                              Section s = new Section(problemsetType);
                              for(Manifest m1 :
children.get(i).getAllDescendantProblemManifests()){
                                    s.belongsTo = children.get(i);
                                    MappingSections.put(m1, s);
                                    s.addChild(m1);
                              }
                        }

                        i++;
                        continue;
                  }

                  // if dangling problems exist
                  ArrayList<Manifest> list = new ArrayList<Manifest>();
                  int j = i;
                  while(i < children.size() && children.get(i).isProblem()){
                        list.add(children.get(i));
                        i++;
                  }
                  Section s = new Section("HomoProblemSet");
                  s.belongsTo = children.get(j); // use first child manifest
as section manifest
                  for(int x = 0; x < list.size(); x++){
                        MappingSections.put(list.get(x), s);
                        s.addChild(list.get(x));
                  }
            }
      }

      @Override
      void setInitialProgressPanelHeader(){
            String display = "Total problems completed: 0";
            view.findSetSectionViewByID(manifest.getUID(), display);
      }

      @Override
      void bookKeeping(Manifest man){
            // used in "Total problems ..." label
            currentRead++;
            // Check if we need to keep the "MoreToCome"
            if(this.manifest.equals(view.getPresenter().getManifest()) ==
true){
                  String display = "Total problems completed: " +
currentRead;
                  view.findSetSectionViewByID(manifest.getUID(), display);
            }

            trackProgress.peek().bookKeeping(man);
            this.moreToCome = moreToCome || trackProgress.peek().moreToCome;
      }
```

```
	@Override
	public void finishAction(){
		this.moreToCome = false;
		if(this.manifest.equals(view.getPresenter().getManifest()) == 
true){
			this.view.removeMoreToCome();
		}
	}

	@Override
	public void setUpHeader(Manifest man){
		checkSection(man);
		if(trackProgress.isEmpty()){
			Section s = MappingSections.get(man);
			ProgressHeaderPresenter newHeader = 
this.chooseHeaderPresenter(s);
			newHeader.manifest = s.belongsTo;
			trackProgress.push(newHeader);
			boolean isSubMixedHeader = newHeader instanceof 
LinearMixedHeaderPresenter || newHeader instanceof RandomMixedHeaderPresenter
					|| newHeader instanceof 
SkillBuilderMixedHeaderPresenter
					|| newHeader instanceof 
IfProblemsetHeaderPresenter
					|| newHeader instanceof 
ChooseProblemsetHeaderPresenter;
			if(isSubMixedHeader){
				newHeader.setUpHeader(man);
			}
			this.moreToCome = moreToCome || newHeader.moreToCome;
			return;
			}

		Section s1 = MappingSections.get(man);
		Manifest m = trackProgress.peek().manifest;
		if(s1.belongsTo.equals(m)){
			boolean isSubMixedHeader = trackProgress.peek() instanceof 
LinearMixedHeaderPresenter || trackProgress.peek() instanceof 
RandomMixedHeaderPresenter
					|| trackProgress.peek() instanceof 
SkillBuilderMixedHeaderPresenter
					|| trackProgress.peek() instanceof 
IfProblemsetHeaderPresenter
					|| trackProgress.peek() instanceof 
ChooseProblemsetHeaderPresenter;
			if(isSubMixedHeader){
				trackProgress.peek().setUpHeader(man);
				this.moreToCome = moreToCome || 
trackProgress.peek().moreToCome;
			}
		}
		else{
			Section s = MappingSections.get(man);
			ProgressHeaderPresenter newHeader = 
this.chooseHeaderPresenter(s);

			newHeader.manifest = s.belongsTo;
```

```
                  trackProgress.push(newHeader);
                  boolean isSubMixedHeader = trackProgress.peek() instanceof
LinearMixedHeaderPresenter || trackProgress.peek() instanceof
RandomMixedHeaderPresenter
                              || trackProgress.peek() instanceof
SkillBuilderMixedHeaderPresenter
                              || trackProgress.peek() instanceof
IfProblemsetHeaderPresenter
                              || trackProgress.peek() instanceof
ChooseProblemsetHeaderPresenter;

                  if(isSubMixedHeader){
                        trackProgress.peek().setUpHeader(man);
                  }
                  this.moreToCome = moreToCome || newHeader.moreToCome;
            }
      }
}
```

org.assistments.gwt.student.tutor.presenter.RandomMixedHeaderPresenter.java

```
package org.assistments.gwt.student.tutor.presenter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

// This is a random problemset containing sections
// composed of mixed headers
public class RandomMixedHeaderPresenter extends ProgressHeaderPresenter{

      private int currentRead = 0;
      HashMap<Manifest, Section> MappingSections; // problem -> Section
      ArrayList<Manifest> danglingProblems;

      public RandomMixedHeaderPresenter(Manifest m, ApplicationView v){
            manifest = m;
            view = v;
            //MoreToCome label should be set to "true" except heading to last
section
            moreToCome = true;
            danglingProblems = new ArrayList<Manifest>();
            MappingSections = new HashMap<Manifest,Section>();
            buildSections(manifest);
            String thisID = manifest.getUID();
            view.addSectionView(thisID);
            //sometimes this presenter will be a sub section, in this case,
the "Total" label should
            //not appear
            if(this.manifest.equals(view.getPresenter().getManifest())){
                  setInitialProgressPanelHeader();
            }
      }
```

```
	/**
	 * no need to build tree, because the order cannot be guaranteed.
	 * only those section which are composed of problems will be have a
	 * basic type header
	 * Otherwise, those problems will be added to danglingProblems
arraylist(the headers will
	 * be added dynamically according to the answering situation)
	 * @param m
	 */
	void buildSections(Manifest m){
		// check if every child is a problem
		// if so no need to build small sections
		List<Manifest> children = m.getChildManifests();
		for(Manifest man : children){
			if(man.isProblemSet()){
				String problemsetType = this.checkManifestType(man);
				Section s = new Section(problemsetType);
				for(Manifest m1 :
man.getAllDescendantProblemManifests()){
					s.belongsTo = man;
					MappingSections.put(m1, s);
					s.addChild(m1);
				}
			}
			else{
				this.danglingProblems.add(man);
			}
		}
	}

	@Override
	void setInitialProgressPanelHeader(){
		String display = "Total problems completed: 0";
		view.findSetSectionViewByID(manifest.getUID(), display);
	}

	@Override
	void bookKeeping(Manifest man){
		currentRead++;
		if(this.manifest.equals(view.getPresenter().getManifest()) ==
true){
			String display = "Total problems completed: " +
currentRead;
			view.findSetSectionViewByID(manifest.getUID(), display);
		}
		trackProgress.peek().bookKeeping(man);
		this.moreToCome = moreToCome || trackProgress.peek().moreToCome;
	}

	@Override
	public void finishAction(){
		this.moreToCome = false;
		if(this.manifest.equals(view.getPresenter().getManifest()) ==
true){
			this.view.removeMoreToCome();
		}
	}

	@Override
	public void setUpHeader(Manifest man){
		checkSection(man);
		if(trackProgress.isEmpty()){
			Section s = MappingSections.get(man);
			if(s != null){

				ProgressHeaderPresenter newHeader = 
this.chooseHeaderPresenter(s);
				newHeader.manifest = s.belongsTo;
				trackProgress.push(newHeader);
				boolean isSubMixedHeader = newHeader instanceof 
LinearMixedHeaderPresenter || newHeader instanceof RandomMixedHeaderPresenter
						|| newHeader instanceof 
SkillBuilderMixedHeaderPresenter
						|| newHeader instanceof 
IfProblemsetHeaderPresenter
						|| newHeader instanceof 
ChooseProblemsetHeaderPresenter;
				if(isSubMixedHeader){
					newHeader.setUpHeader(man);
				}
				this.moreToCome = moreToCome || newHeader.moreToCome;
			return;
		}

			//man is a dangling problem
			IndeterminedHeaderPresenter newheader = new 
IndeterminedHeaderPresenter(man,view);
			trackProgress.push(newheader);
			this.moreToCome = moreToCome || 
trackProgress.peek().moreToCome;
			return;
		}

		// m belongs to a section s1, 2 cases
		// 1:s1 is on the top on the stack, only need to bookkeeping that 
top element
		// 2: s1 is not on the stack, need to create a header presenter 
for that section
		Section s1 = MappingSections.get(man);
		if(s1 != null){
			Manifest m = trackProgress.peek().manifest;
			if(s1.belongsTo.equals(m)){
				boolean isSubMixedHeader = trackProgress.peek() 
instanceof LinearMixedHeaderPresenter || trackProgress.peek() instanceof 
RandomMixedHeaderPresenter
						|| trackProgress.peek() instanceof 
SkillBuilderMixedHeaderPresenter
						|| trackProgress.peek() instanceof 
IfProblemsetHeaderPresenter
						|| trackProgress.peek() instanceof 
ChooseProblemsetHeaderPresenter;
				if(isSubMixedHeader){
					trackProgress.peek().setUpHeader(man);
```

```
                              this.moreToCome = moreToCome ||
trackProgress.peek().moreToCome;
                        }
                        return;
                  }
                  //s1 is a new section, need to build a new header for it
                  else{
                        ProgressHeaderPresenter
newHeader=this.chooseHeaderPresenter(s1);
                        newHeader.manifest = s1.belongsTo;
                        trackProgress.push(newHeader);
                        boolean isSubMixedHeader = trackProgress.peek()
instanceof LinearMixedHeaderPresenter || trackProgress.peek() instanceof
RandomMixedHeaderPresenter
                                    || trackProgress.peek() instanceof
SkillBuilderMixedHeaderPresenter;
                        if(isSubMixedHeader){
                              trackProgress.peek().setUpHeader(man);
                        }
                        this.moreToCome = moreToCome || newHeader.moreToCome;
                        return;
                  }
            }
            // m does not belong to a specific section,
            // then it is a dangling problem,2 cases
            // 1. previously having a indetermined header presenter, update
that one
            // 2. no indetermined header presenter, create one.
            else{
                  if(this.danglingProblems.contains(man)){

      if(this.danglingProblems.contains(trackProgress.peek().manifest)){
                              return;
                        }
                        // if top of stack is a section, then we need to
build a new header
                        IndeterminedHeaderPresenter newheader = new
IndeterminedHeaderPresenter(man,view);
                        trackProgress.push(newheader);
                  }
            }
      }
}
```

org.assistments.gwt.student.tutor.presenter.SkillBuilderMixedHeaderPresenter.java

```
skillbuildermpackage org.assistments.gwt.student.tutor.presenter;

import org.assistments.core.domain.PropertyKeys;
import org.assistments.core.domain.content.Manifest;
import org.assistments.gwt.student.tutor.view.ApplicationView;

public class SkillBuilderMixedHeaderPresenter extends
ProgressHeaderPresenter{

      int masterProblems;
      public SkillBuilderMixedHeaderPresenter(Manifest m, ApplicationView v){
            manifest = m;
            view = v;
            masterProblems =
Integer.parseInt(manifest.getProperties().get(PropertyKeys.TERMINATOR_MASTERY
LIMIT));
            moreToCome = true;
            String thisID = manifest.getUID();
            view.addSectionView(thisID);
            setInitialProgressPanelHeader();
      }

      @Override
      void setInitialProgressPanelHeader(){
            String display = "Answer " + masterProblems + " correctly in a
row";
            view.findSetSectionViewByID(manifest.getUID(), display);
            //if it's a simple skill builder, we need to stwitch off the
moreToCome
            //Or, we should consider it to be a "uncertain" section, similar
to LinearMixed
            //and RandomMixed
            if(this.manifest.equals(this.view.getPresenter().getManifest())){
                  this.moreToCome = false;
            }
      }

      @Override
      void bookKeeping(Manifest man){
            // TODO Auto-generated method stub

      }

      @Override
      public void finishAction(){
            this.moreToCome = false;
            if(this.manifest.equals(view.getPresenter().getManifest()) ==
true){
                  this.view.removeMoreToCome();
            }

      }

      @Override
      public void setUpHeader(Manifest man){
            // TODO Auto-generated method stub

      }

}
```

org.assistments.gwt.sudent.tutor.view.ApplicationView.java

```
/*****************************************************************************
***
 * Copyright (c) 2010, 2011 Worcester Polytechnic Institute
 * All rights reserved.

*****************************************************************************
**/
package org.assistments.gwt.student.tutor.view;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.assistments.gwt.core.BuildInfo;
import org.assistments.gwt.core.client.resources.ApplicationResources;
import org.assistments.gwt.core.client.view.DeveloperPasswordUtil;
import org.assistments.gwt.student.tutor.domain.ProgressPanelResponseTypes;
import org.assistments.gwt.student.tutor.presenter.ApplicationPresenter;
import org.assistments.gwt.student.tutor.presenter.ProgressPanelIcon;
import
org.assistments.gwt.student.tutor.view.devconsole.DeveloperConsoleView;
import org.assistments.util.StringCleaner;

import com.google.gwt.core.client.GWT;
import com.google.gwt.dom.client.Style;
import com.google.gwt.dom.client.Style.Display;
import com.google.gwt.dom.client.Style.FontWeight;
import com.google.gwt.dom.client.Style.TextDecoration;
import com.google.gwt.dom.client.Style.Unit;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.event.dom.client.KeyPressEvent;
import com.google.gwt.event.dom.client.KeyPressHandler;
import com.google.gwt.event.dom.client.MouseOutEvent;
import com.google.gwt.event.dom.client.MouseOutHandler;
import com.google.gwt.event.dom.client.MouseOverEvent;
import com.google.gwt.event.dom.client.MouseOverHandler;
import com.google.gwt.uibinder.client.UiBinder;
import com.google.gwt.uibinder.client.UiConstructor;
import com.google.gwt.uibinder.client.UiField;
import com.google.gwt.uibinder.client.UiHandler;
import com.google.gwt.user.client.Timer;
import com.google.gwt.user.client.ui.Anchor;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.DecoratedPopupPanel;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.DisclosurePanel;
import com.google.gwt.user.client.ui.DockLayoutPanel;
import com.google.gwt.user.client.ui.FlowPanel;
import com.google.gwt.user.client.ui.FocusPanel;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.HTMLPanel;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Image;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.LayoutPanel;
import com.google.gwt.user.client.ui.SplitLayoutPanel;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;
import com.google.inject.Inject;

public class ApplicationView extends Composite implements KeyPressHandler {
  private static ApplicationViewUiBinder uiBinder =
    GWT.create(ApplicationViewUiBinder.class);

  interface ApplicationViewUiBinder extends UiBinder<Widget,ApplicationView>
{
  }

  ApplicationPresenter presenter;

  @UiField LayoutPanel assignmentContainer;
  @UiField FlowPanel progressPanel;
  @UiField FocusPanel focusPanel;
  @UiField SplitLayoutPanel splitPanel;
  @UiField HTML betaNotice;
  @UiField HTMLPanel accountInfo;
  @UiField HorizontalPanel headerTabs;
  @UiField HTMLPanel headerStyle;
  @UiField Anchor assignmentsLink;
  @UiField HTMLPanel indicatorField;

  //@UiField(provided=true) DeveloperConsoleView developerConsole;
  DeveloperConsoleView developerConsole;

  @UiField DockLayoutPanel dockPanel;

  @UiField HTMLPanel problemsCompleteHTML;
  @UiField HTMLPanel affectDetectorHTML;
  Label problemsCompleteLabel;

  // Map: Manifest -> Manifest ID -> SectionHTML
  Map<String, SectionHTML> sectionToHtmls;
  // "More to come..." header / label
  SectionHTML moreToCome;

  // Every problem panel has a popup for when you hover over it.  We keep
this map so we can
  // have references to those popups if we need to change the text in it.
  private Map<DisclosurePanel, DecoratedPopupPanel> popups;

  private PopupTimer timer = new PopupTimer();
  private ApplicationResources appResources;

  /**
   * This class defines each section header's view
   *
   */
  class SectionHTML
  {
    private HTMLPanel htmlPanel;
    private Label sectionLabel;
    private boolean hasDisplayed;

    public SectionHTML()
    {
      htmlPanel = createAndStyleProgressPanelHeader();
      hasDisplayed = false;
      sectionLabel = new Label();
      htmlPanel.add(sectionLabel);
    }

    private HTMLPanel createAndStyleProgressPanelHeader()
    {
      HTMLPanel header = new HTMLPanel("");
      header.getElement().getStyle().setMarginTop(10, Unit.PX);
      header.getElement().getStyle().setMarginBottom(5, Unit.PX);
      header.getElement().getStyle().setBackgroundColor("#005192");
      header.getElement().getStyle().setFontWeight(FontWeight.BOLD);
      header.getElement().getStyle().setColor("White");
      header.getElement().getStyle().setPadding(5, Unit.PX);

header.getElement().setClassName(appResources.style().problemsComplete());
      header.setWidth("85%");

      return header;
    }

    public SectionHTML(String label)
    {
      this();
      this.setText(label);
    }

    public HTMLPanel getPanel()
    {
      return htmlPanel;
    }

    public boolean isHasDisplayed( )
    {
      return hasDisplayed;
    }

    public void setHasDisplayed(boolean hasDisplayed)
    {
      this.hasDisplayed = hasDisplayed;
    }

    public void setText(String str)
    {
      sectionLabel.setText(str);
    }
  }

  @Inject
  @UiConstructor
  public ApplicationView(ApplicationResources appResources,
      DeveloperConsoleView developerConsole) {
    this.appResources = appResources;
    this.developerConsole = developerConsole;

    this.developerConsole.setParentView(this);
    initWidget(uiBinder.createAndBindUi(this));
    popups = new HashMap<DisclosurePanel, DecoratedPopupPanel>();

    /**AS-951*/
    this.sectionToHtmls = new HashMap<String,SectionHTML>();
    moreToCome = new SectionHTML("More to come...");
   /***/

    progressPanel.setWidth("95%");
    progressPanel.getElement().getStyle().setMarginTop(10, Unit.PX);
    progressPanel.getElement().getStyle().setMarginLeft(10, Unit.PX);
    //At minimum, we need to display the tutor tab for users to exit the new
tutor
    headerTabs.add(formatHTMLRole("tutor", "Student", true));

    focusPanel.addKeyPressHandler(this);
    progressPanel.setVisible(false);

    // To enable the Beta notice, edit BuildInfo.java.template: IS_BETA
    if(BuildInfo.isBeta())
    {
      Style betaStyle = betaNotice.getElement().getStyle();
      betaStyle.setDisplay(Display.BLOCK);
    }

    clearBlankScreenInfo();

    // Keeping this here for easy access in future. Uncomment this and
    // the tutor will load with the developer console showing right away.
    //DeveloperConsoleView.toggleDeveloperConsole();
  }

  public static native void clearBlankScreenInfo() /*-{
      var infoDiv = $doc.getElementById("whitePageInfo");
      infoDiv.style.visibility="hidden";
}-*/;

  public void showDevConsole() {
    splitPanel.clear();
    splitPanel.addSouth(developerConsole, 300);
    splitPanel.add(dockPanel);
  }

  public void hideDevConsole() {
    splitPanel.clear();
    splitPanel.add(dockPanel);
  }

  public void setAccountName(String displayName, String loginName) {
    String displayString = displayName  + " (" + loginName + ")";
    Label userNameLabel = new Label(displayString);
    accountInfo.add(userNameLabel,"accountName");
  }

  public void setRoleAllowances(List<String> roles){
    if (roles.size() != 0) {
      headerTabs.clear();

headerStyle.getElementById("headerBar").setClassName(appResources.style().hea
derBar());
      if (roles.contains("Teacher")){          // Teacher tab
        headerTabs.add(formatHTMLRole("teacher", "Teacher", false)); }
      if (roles.contains("Student")){          // Student tab
        headerTabs.add(formatHTMLRole("tutor", "Student", true)); }
      if (roles.contains("ContentCreator")){       // Builder tab
        headerTabs.add(formatHTMLRole("build", "Builder", false)); }
      if (roles.contains("Administrator")){      // Admin tab
        headerTabs.add(formatHTMLRole("admin", "Admin", false));

headerStyle.getElementById("bannerContent").setClassName(appResources.style()
.adminBanner());

headerStyle.getElementById("headerBar").setClassName(appResources.style().adm
inHeaderBar()); }
      if (roles.contains("Researcher")){         // Develop tab
        headerTabs.add(formatHTMLRole("researcher", "Research", false));  }
      if (roles.contains("Developer")){        // Develop tab
        headerTabs.add(formatHTMLRole("developer", "Develop", false));  }
      if (roles.contains("Parent")){           // Notify tab
        headerTabs.add(formatHTMLRole("parent", "Notify", false));  }
      if (roles.contains("DistrictAdministrator")){ // District tab
        headerTabs.add(formatHTMLRole("district_admin", "District", false));
}
      if (roles.contains("SchoolAdministrator")){   // School tab
        headerTabs.add(formatHTMLRole("school_admin", "School", false));  }
      if (roles.contains("StateAdministrator")){    // State tab
        headerTabs.add(formatHTMLRole("state_admin", "State", false));  }
    }
    /* Granted roles without access to a specific tab
    if ("Staff"){}
    if ("SchoolFolder"){}
    if ("DistrictFolder"){}
    if ("StateFolder"){}*/
  }

  /**
   * Creates an HTML tab to be used in the header bar
   * @param title The literal text on the tab
   * @param href The href relative text to get to the link
   * @param hilighted Whether this tab is the highlighted tab or not
   * @return The properly formatted HTML tag
   */
  private HTML formatHTMLRole(String href, String title, boolean hilighted){
    if (hilighted){
      return new HTML("<a href=\"/" + href + "\" id=\"hilightedHeaderTab\">"
+ title + "</a>");
    } else {
      return new HTML("<a href=\"/" + href + "\" id=\"commonHeaderTab\">" +
title + "</a>");
    }
  }

  @Override
  public void onKeyPress(KeyPressEvent event) {
    if (DeveloperPasswordUtil.getInstance().recordKeyPressEvent(event)) {
      DeveloperConsoleView.toggleDeveloperConsole();
    }
  }

  public LayoutPanel getAssignmentContainer() {
    return assignmentContainer;
  }

  public void addToAssignmentPanel(Widget w) {
    assignmentContainer.clear();
    assignmentContainer.add(w);
  }

  /**
   * If active is true, then add a blue arrow image to the left of the
disclosure panel.
   * @param panel
   * @param active
   */
  public void markItemActive(DisclosurePanel panel, boolean active) {
    HorizontalPanel hpanel = (HorizontalPanel)panel.getParent();
    hpanel.remove(0);
    if (active) {
      hpanel.insert(new Image(appResources.rightArrow()),0);
    } else {
      HTML blank = new HTML();

blank.getElement().getStyle().setWidth(appResources.rightArrow().getWidth(),
Unit.PX);
      hpanel.insert(blank, 0);
    }
  }

  // dm TODO: Again, it seems the Progress Panel should have its own
presenter and view.
  /**
   * Add a new problem to the progress panel.
   * @param manifestKey The Manifest's persistence key
   * @param assistmentID The Problem's ID. Ideally this is the encoded
assistment ID (ex: PRABCD),
   * but lacking that, send in the actual problem ID.
   * @param questionHtml The problem's question. This is most likely going to
be html decorated
   * <tt>questionHtml</tt> is used in two ways: (a) to create a title for the
panel entry and (b)
   * to be displayed in full in a pop-up on a mouse-over.
   * @param defaultTitle If <tt>questionHtml</tt> contains no usable text
(possible if for example
   * the entire question is composed of one or more image(s), use this as the
title text.
   * @return the progress panel
   */
  public DisclosurePanel addProblem(final String manifestKey, String
assistmentID, String questionHtml,
      String defaultTitle)
  {
    HorizontalPanel rootpanel = new HorizontalPanel();
    HTML blank = new HTML();

blank.getElement().getStyle().setWidth(appResources.rightArrow().getWidth(),
Unit.PX);
    rootpanel.add(blank);

    String title = this.getTitleString(questionHtml, defaultTitle);

    final DisclosurePanel dpanel = new DisclosurePanel(title);
    final DecoratedPopupPanel popup = new DecoratedPopupPanel(true);
    popups.put(dpanel, popup);

    dpanel.addDomHandler(new MouseOverHandler() {
      @Override
      public void onMouseOver(MouseOverEvent event) {
        Widget source = (Widget)event.getSource();
        popup.setPopupPosition(source.getAbsoluteLeft() + 165,
source.getAbsoluteTop());

dpanel.getHeader().getElement().getStyle().setTextDecoration(TextDecoration.U
NDERLINE);
        // Start the timer to show the popup.
        timer.setupTimer(manifestKey, popup);
      }
    }, MouseOverEvent.getType());

    dpanel.addDomHandler(new MouseOutHandler() {
      @Override
      public void onMouseOut(MouseOutEvent event) {

dpanel.getHeader().getElement().getStyle().setTextDecoration(TextDecoration.N
ONE);
        popup.hide();
        timer.cancel();
      }
    }, MouseOutEvent.getType());

    dpanel.add(new StudentResponsesView(appResources));
    rootpanel.add(dpanel);
    progressPanel.add(rootpanel);

    setPanelText(assistmentID, dpanel, questionHtml, defaultTitle);
    return dpanel;
  }

  public String getTitleString(String source, String defaultTitle)
  {
    String title = defaultTitle;
    String cleanedSource = StringCleaner.truncateString(source,19);

    if(cleanedSource.length() > 0)
    {
      title = cleanedSource;
    }

    return title;
  }

  /**
   * Change the display text of a given DisclosurePanel.
   * @param panel
   * @param body
   */
  public void setPanelText(String assistmentID, DisclosurePanel panel, String
body, String defaultTitle) {
    progressPanel.setVisible(true);

    String title = this.getTitleString(body, defaultTitle);

    panel.setHeader(new HTML(title));

    HTML html = new HTML("<span style=\"color:#666666;padding-
left:10px;padding-right:10px;font-style:italic;\">Problem ID: <span
style=\"font-weight:bold;\">" +
            assistmentID +
            "</span></span><br/>" +
            body);
    popups.get(panel).setWidget(html);
  }

  public void setPopupLoading(DisclosurePanel panel) {
    HorizontalPanel hp = new HorizontalPanel();
    hp.add(new Label("Loading problem..."));
    Image img = new Image(appResources.indicator());
    img.getElement().getStyle().setMarginLeft(10, Unit.PX);
    hp.add(img);
    popups.get(panel).setWidget(hp);
  }

  /**
   * Add an action that does not specify/show correctness, such as for test
mode (defaults to true).
   * @param panel
   * @param answer
   * @param seconds
   * @param trunc
   */
  public void addAction(DisclosurePanel panel, String answer, int seconds,
ProgressPanelResponseTypes responseTypes, boolean trunc) {
        if(trunc == true){

((StudentResponsesView)panel.getContent()).addResponse(StringCleaner.truncate
String(answer, 12), seconds, responseTypes);
        }
        else{
              ((StudentResponsesView)panel.getContent()).addResponse(answer,
seconds, responseTypes);
        }
  }

  public void setProblemsComplete(int complete, int total) {
    problemsCompleteLabel.setText("Problem: " + complete + " / " + total);
  }
  public void setProblemsCompleteLabel(String label){
        problemsCompleteLabel.setText(label);
  }

  public void setProblemsComplete(int complete){
        problemsCompleteLabel.setText("Problems completed: " + complete);
  }

  public void setProblemsCorrectInARow(int correct, int total) {
    problemsCompleteLabel.setText("Correct in a row: " + correct + " -
Target: " + total);
  }

  /**
   * Clears the panel containing the problems.
   * Leaves the Progress Panel displayed.
   * Leaves the DevConsole available.
   */
  public void clearAssignmentContainer() {
    assignmentContainer.clear();
  }

  public void hideAllButDevConsole()
  {
    // dm: leaving past commented out approaches (in case they prove
    // useful in the future)
    //  progressPanel.removeFromParent();
    //  assignmentContainer.removeFromParent();

    progressPanel.clear();
    assignmentContainer.clear();
}

  /**
   * Simple time class which just shows a PopupPanel after a delay.
   */
  private class PopupTimer extends Timer {
    private static final int POPUP_DELAY_MS = 650;
    private String manifestKey;
    private DecoratedPopupPanel popup;

    public void setupTimer(String manifestKey, DecoratedPopupPanel popup) {
      this.popup = popup;
      this.manifestKey = manifestKey;
      this.schedule(POPUP_DELAY_MS);
    }
    @Override
    public void run() {
      presenter.requestContent(manifestKey);
      popup.show();
    }
  }

  // dm: AS-876
  // Show the "Assignments" link when requested.
  public void showAssignmentsLink()
  {
    assignmentsLink.setVisible(true);
  }

  // dm: AS-876
  // Disable all buttons and show an indicator when the link is clicked.
  @UiHandler("assignmentsLink")
  void onAssignmentsLinkClick(ClickEvent evt)
  {
    Image img = new Image(appResources.indicator());
    indicatorField.add(img);
    indicatorField.setVisible(true);
    presenter.onAssignmentListLinkClicked();
  }

  @UiHandler("aboutLink")
  void onClick(ClickEvent evt) {
    final DialogBox box = new DialogBox();

    box.setText("About");
    VerticalPanel panel = new VerticalPanel();
    panel.add(new AboutView());
    Button closeButton = new Button("Close",
      new ClickHandler() {
        @Override
        public void onClick(ClickEvent evt) {
          box.hide();
          // dm: AS_1013: Decided to glass the About dialog.
          // So need to clear it here.
          box.setGlassEnabled(false);
        }
      }
    );
    panel.add(closeButton);
    box.setWidget(panel);

    // dm: AS_1013: Decided to glass the About dialog
    // while messing with z-indexes.
    box.setModal(false);
    box.setGlassEnabled(true);
    box.center();
  }

      @UiHandler("settingsLink")
      void onSettingsLinkClicked(ClickEvent evt) {
             presenter.showSettings();
      }

  public void setPresenter(ApplicationPresenter presenter) {
    this.presenter = presenter;
```

```
}
public ApplicationPresenter getPresenter() {
  return this.presenter;
}

/**
 * add the label passed down to the HTML label
 * @param problemSetUpdateLabel
 * @aurora
 */
public void addAffectDetectorPanel(AffectDetectorView adview) {
  affectDetectorHTML.add(adview);
}

public void clearProgressPanel() {
  this.progressPanel.clear();
}

/**
 * create a new SectionHTML and add it into the map
 * @param s
 */
public void addSectionView(String s){
      SectionHTML shtml = new SectionHTML();
      sectionToHtmls.put(s, shtml);
}
/**
 * Find the corresponding view by its manifest ID
 * and set the text as the parameter display
 * @param ID
 * @param display
 */
public void findSetSectionViewByID(String ID, String display){
      SectionHTML sh = sectionToHtmls.get(ID);
      if(sh == null){
            return;
      }
      if(sh.isHasDisplayed() == false){
            this.progressPanel.add(sh.getPanel());
            //this.progressPanel.add(sh.htmlPanel);
            sh.setHasDisplayed(true);
      }
      sh.setText(display);
}
/**
 * display the "MoreToCome" Label
 */
public void showMoreToCome(){
      if(moreToCome.isHasDisplayed() == false){
            this.progressPanel.add(moreToCome.getPanel());
      }
      else{
            this.progressPanel.remove(moreToCome.getPanel());
            this.progressPanel.add(moreToCome.getPanel());
  }
  moreToCome.setHasDisplayed(true);
}
```

```
  /**
   * remove the "MoreToCome" Label from progresspanel
   */
  public void removeMoreToCome(){
    this.progressPanel.remove(moreToCome.getPanel());
    moreToCome.setHasDisplayed(false);
  }

}
```