Implementation Readiness Package (IRP v2)

Developer's Guide

Prepared by UCLA/CRESST

Table of Contents

Setup

This section describes the build process for IRP.

Dependencies

- Java 7 JDK
- Maven 3.2 and above – Used as package manager, dependency manager, and build tool
- irp-package – The irp-package repository contains IRP data. This is a separate repository and should exist on the same development machine as the IRP source code. It is located at https://bitbucket.org/sbcresst/irp-package.
- node/npm, bower, bower-installer – For Polymer based user interface
- An Item Scoring Engine – Used to perform Item Scoring Analysis. The one used for development is the Smarter Balanced Open Source Item Scoring Engine (https://github.com/SmarterApp/TDS_ItemScoring) hosted at http://tds.smarterapp.cresst.net:8080/item-scoring-service/Scoring/ItemScoring.
- The Test Integration System – Used to perform Test Scoring Analysis. The one used for development is the Smarter Balanced Open Source Test Integration System (https://github.com/SmarterApp/TDS_TestIntegrationSystem) hosted at http://tis.smarterapp.cresst.net/oss_tisservices/api/TestResult.

Configuration

See irp-webapp/src/main/resources/irp.properties

Configuration can be done either by editing this file or specifying environment variables

Environment Variable	Use
ITEM_SCORING_SERVICE_URL	URL to the Item Scoring Service. IRP uses its own instance of an Item Scoring Service to calculate Item Scores to compare against the Item Scores in the TDS Report XML.
TEST_SCORING_SERVICE_URL	URL to the Test Scoring Service. IRP uses its own instance of a Test Scoring Service to calculate Test Scores to compare against the Test Scores in the TDS Report XML.
IRP_PACKAGE_LOCATION	Specify the location of the IRP Package. Note: schema is REQUIRED. E.g.: file:///path/to/irp/package
CAT_DATAFOLDER	Specify the location of the folder for CAT Analysis module. Note: This only accepts local files system paths and a schema should NOT be specified.

Building

First, ensure that the development machine's environment variables or irp.properties file point to valid paths.

Maven is used to build IRP. Build with `mvn clean install` from the root of the repository.

Running

First, ensure that the development machine's environment variables or irp.properties file point to valid paths.

Run with `mvn tomcat7:run` at the root of the repository. This will start the irp-webapp.

Module Overview

irp-core

org.cresst.sb.irp.exceptions

Contains exceptions used in all the IRP modules. These include: *NotFoundException*, for resources which cannot be found; *UnhandledException*, exceptions which are not handled; *ValidationException*, for validation errors which occur in creating resources.

irp-model

This module consists of all the java representations of the Smarter Balanced data. Consists of representations of the XML data objects. Contains the following:

- org.cresst.sb.irp.domain.accommodation
- org.cresst.sb.irp.domain.items
- org.cresst.sb.irp.domain.manifest
- org.cresst.sb.irp.domain.student
- org.cresst.sb.irp.domain.tdsreport
- org.cresst.sb.irp.domain.testpackage
- org.cresst.sb.irp.domain.teststudentmapping

irp-service

This module consists of all the spring services. For each service, there is an interface <Service Name>Service.java and a corresponding implementation called <Service Name>ServiceImpl.java. Each of services provide a wrapper around the corresponding DAO repository found in **irp-repository**. For example, ItemService provides functionality to access items in irp-repository: org.cresst.sb.irp.dao.ItemDao

The services in **org.cresst.sb.irp.service** available are:

- *AccommodationService*: Enables access to accommodations by Student Identifier (SSID).
- *ItemService*: Enable retrieving items by fields: id, identifier; Getting properties of items.
- *ManifestService*: Retrieve manifests by identifier; get lists of manifest resources, access metadata, and get organizations string.
- *StudentService*: Retrieve students by SSID, create student in the StudentDAO, get a list of all students.
- *TDSReportService*: Retrieve test, examinee, opportunity, list of all opportunity scores, list of all opportunity items, list of all comments, list of all the tool usage.
- *TestPackageService*: Get map of test packages, get test package by unique id, get various properties about the test packages
- *TestStudentMapping*: Get a list of test student mappings, get a specific test student mapping by specifying test name and student SSID.

irp-repository

org.cresst.sb.irp.dao

This module includes all the Data Access Objects (DAO) in the form of spring repositories. Each repository has an interface with a corresponding implementation.

- Accommodation
- Item
- Manifest
- Student
- TDSReport
- TestPackage
- TestStudentMapping

org.cresst.sb.irp.utils

This package provides functionality for the DAO access to parse and process data. There are specific utilities for some of the DAOs and some generic utilities like ExcelUtil, RetrieveFileUtil, and XMLValidate.

irp-tdsreport-analysis-engine

The core module of the IRP analysis. Provides the functionality to process TDS reports from XML files, score items, score tests, analyze fields, and cell categories. All the processing starts from *org.cresst.sb.irp.service.AnalysisService.java* where a single *org.cresst.sb.irp.domain.analysis.AnalysisResponse.java* is produced.

org.cresst.sb.irp.analysis.engine

Represents a way to analyze a category from the Smarter Balanced specifications. Uses both

- Data Model: http://www.smarterapp.org/documents/TestResults-DataModel.pdf
- Test Results Transmission Format (TRT): http://www.smarterapp.org/documents/TestResultsTransmissionFormat.pdf

In case of a discrepancy between the Data Model and the TRT, the Data Model is used.

The main file in this package is the abstract class: AnalysisAction.java. All other classes in this package implement this abstract class to provide specific implementations for the various Smarter Balanced categories. For example: TestAnalysisAction.java provides the functionality to analyze a Test Analysis section of the TDS report. Some of the common features across all the Analysis Actions are

- An enum representing the keys in the category. Additional information is provided, through constructors:
  - max width: maximum number of characters allowed in the field
  - required boolean: If this field is optional then the constructor should provide "false" for the required status. Defaults to required if not specified.

org.cresst.sb.irp.domain.analysis

Contains all the representations of a category, an individual's response, field check type, opportunity item data,

org.cresst.sb.irp.itemscoring

Contains all the item scoring functionality.

- IrpProxyItemScorer.java: gets item score from Item Scoring Service
- ItemScoreMessageConverter.java: Converts Item Scoring Service requests and responses to ItemScoreRequest and ItemScoreResponse objects, respectively.
- MachineRubricFileSystemLoader.java: Loads the contents of machine rubrics from the file system.

org.cresst.sb.irp.service

Provides a service, Analysis Service, which analyzes TDS reports to produce a final response.

org.cresst.sb.irp.testscoring

Implements test scoring functionality for a given TDS report.

- TestIntegrationSystemScorer.java: Calls the Test Integration System (TIS) service to score a TDSReport

irp-cat-analysis-engine

The CAT Module of IRP provides 4 packages. The core analysis begins in org.cresst.sb.irp.cat.service, which provides a function for analyzing CAT data from a CATDataModel (found in org.cresst.sb.irp.cat.domain.analysis.CATDataModel) and results in a org.cresst.sb.irp.cat.domain.analysis.CATAnalysisResponse. To see how to create a CATDataModel and analyze CAT Data see irp-webapp/org.cresst.sb.irp.upload.CATFileUploadController.java

The CAT Analysis module of IRP runs with the following steps:

- The vendor downloads a "Simulation package", which includes all the information needed to run the simulation. This package includes: the item pool, simulated responses, and blueprint specifications.
- The vendor uses this information to run a simulation using their own CAT engine.
- Once the vendor's CAT simulation is complete, they will upload their results to IRP for evaluation. The uploaded data must include two components:
    a. **Items Datafile**: a record of every item administered to each simulees (as selected by the test delivery system's adaptive algorithm)
    b. **Student Scores Datafile**: final scale score estimates obtained and their associated standard errors of measurement.

  Datafiles corresponding to these two components must be delivered in a comma-separated format.
- Once the uploaded data file is received from the vendor, IRP will examine the file to verify that its format conforms to the fields and data types described in Data File Specifications found below. If there are any errors detected, IRP will flag them and issue a report, and the Data Evaluation process will end. If no errors are found, IRP will proceed with the Data Evaluation process.

org.cresst.sb.irp.cat.analysis.engine

Provides a service, *CATParsingService.java*, which takes CAT Data files (specified in Data Files section) and produces lists of objects representing either Items, Students, Pool Items, True Theta values, or blueprint rows. All of these objects are found in domain.analysis below. CSV files are parsed using Jackson-csv parser.

org.cresst.sb.irp.cat.domain.analysis

Contains classes representing the rows from the csv data files and classes needed for analysis. Each of the csv rows are specified with Jackson-csv syntax. The header of the csv is represented in the *JsonPropertyOrder* annotation. The files representing rows in a csv object are the following:

- *TrueTheta.java*
- *ThresholdLevels.java*
- *PoolItemMath.java*
- *PoolItemELA.java*
- *ItemResponseCAT.java*
- *ELAStudentScoreCAT.java*
- *MathStudentScoreCAT.java*
- *BlueprintCsvRow.java*

Other classes include:

- BlueprintCondition.java: Provides an interface to test a pool item if it matches a given specification. This helps in the creation of blueprints as we can use anonymous inline classes to specify the test condition when creating a blueprint.
- BlueprintStatement.java: The results of processing a BlueprintCsvRow. This processing is done in org.cresst.sb.irp.cat.analysis.engine.CATParsingServiceImpl.parseBlueprintCsv. The additional functionality a BlueprintStatement has over a BlueprintCsvRow is a test function, as described above; a ViolationCount (org.cresst.sb.irp.cat.domain.analysis.ViolationCount), specification string, and functions to update the matches for a given blueprint statement.
- CATAnalysisResponse.java: Results of analyzing CAT Data Model. Contains results
    - item exposure rate
    - average bias
    - root mean squared error (RMSE)
    - blueprint statements
    - standard error of measurement (SEM)
    - claim violations
    - classification levels
- *CATDataModel.java*: Contains data needed for analysis
    - Student scores
    - Item responses
    - Pool items
    - True thetas
    - Blueprint statements

    - Grade
    - Subject
- Other analysis files include: ExposureRate.java, ExposureRateResponse.java, ViolationCount.java

org.cresst.sb.irp.cat.service

Provides the main service for CAT Data Analysis, CATAnalysisService.java. All analysis starts in function analyzeCatData.

org.cresst.sb.irp.cat.service.lib

Contains one library, Stats.java, which provides static methods for calculating average bias, mean squared error, root mean squared error, calculating exposure rates, calculating exposure rate bins, decile partitioning, score level matrix based on theta cutoff levels, and standard error measurement calculations.

irp-webapp

The main web application for IRP. Provides the functionality to run manual mode IRP from xml TDS reports, automation mode to get TDS reports from a vendor's adapter, and CAT Data Analysis.

org.cresst.sb.irp.automation

This contains a controller to processes/analyzes TDS Report XMLs located at URIs provided in an AdapterData object. This is called by IRP's UI in Automation Mode. The controller's endpoint is /analysisReports.

org.cresst.sb.irp.cat

- ResourceSelector.java: Returns resources from CAT Data Folder
- SimulationPackage.java: Spring Controller with endpoints for downloading blueprints, item pool, and student response data.
- CATPDFController.java: Spring Controller to download a PDF version of the CAT Data Analysis.

org.cresst.sb.irp.download

Provides endpoint for downloading irp-package folder as a zip.

org.cresst.sb.irp.exception

Provides an exception when IRP is unable to generate a PDF.

org.cresst.sb.irp.index

Provides route to get IRP version number.

org.cresst.sb.irp.jsondoc

Provides route to view jsondoc API documentation.

org.cresst.sb.irp.report

Provides endpoint and methods for generating a PDF version of the analysis report.

org.cresst.sb.irp.rest

RESTful API routes for accessing Items, Manifest, Students, TDS Reports, and Test Packages.

org.cresst.sb.irp.upload

Handles HTTP-POST calls to upload data files to IRP for both all modules in IRP.

org.cresst.sb.irp.zip

Utilities for zipping files.

User Interface

The user interface is developed mainly using Polymer Project 1.0 (https://www.polymer-project.org/1.0/). The main entry point to IRP's user interface is through `irp-webapp/webapp/index.html`. Note, all of the user interface components exist in the `irp-webapp/webapp` directory. Furthermore, the user interface is driven by the JavaScript file `scripts/app.js`.

The main entry point, `index.html`, is composed of custom IRP elements and third-party elements. Custom IRP elements are located in the `elements` directory. Third-party elements are located in the `deps` directory. The `elements/elements.html` file acts as a single import statement. It contains links to all the relevant elements, both custom IRP elements and third-part elements.

Third-party elements are committed to source code in the `deps` directory. Bower and bower-installer are used to install the third-party elements into the deps directory using the following command:

```
# bower-installer --save <Third-party element>

# bower-installer --save PolymerElements/paper-toast
```

In the example above, the paper-toast element is saved to the `deps` directory. Afterwards, the new files are committed to the source code repository. However, to use bower-installer effectively, the `bower.json` file will need to be modified to include the files from the third-party element's list of available files. Typically, a third-party element will come with extraneous files. Use the `bower.json` file to specify only the production files that are necessary to use the element.

Endpoints

POST /analysisReports produces "json/AnalysisResponse"

Content Type: application/json

Request Body: AdapterData (json)

GET /simupack/blueprints

attachment download of the blueprints csv file.

Content Type: text/csv

GET /simupack/itempool/subject/{subject}

Attachment download of the CAT item pool for {subject}

Content Type: text/csv

GET /simupack/studentdata/subject/{subject}/grade/{grade}

Attachment download of the CAT student data for {subject} and {grade}

Content Type: text/csv

POST /pdfreport.html

Download a PDF representation of the IRP TDS report analysis

Content Type: "application/x-download"

Request Parameter: analysisResponses (json representation of org.cresst.sb.irp.domain.analysis.AnalysisResponse)

POST /catPdfreport.html

Download a PDF representation of the CAT Analysis results

Content Type: "application/x-download"

Request Parameter: catResults (json representation of org.cresst.sb.irp.cat.domain.analysis.CATAnalysisResponse)

GET /irp-package.zip

Downloads the irp-package

Content Type: application/zip

POST /catUpload/subject/{subject}/grade/{grade}

Analyzes CAT Data and produces an analysis result.

Content Type: application/json (CATAnalysisResponse)

Request Parameters:

- catItemFile: MultipartFile of the item data.
- catStudentFile: MultipartFile of the student data.
- vendorName: (Optional) string of the vendor

GET /irp-version.html

Get the current running version of IRP

Content Type: html

REST API

There is a RESTful API for the providing access to items, manifests, students, TDS reports, and test packages. The documentation is available on the route (using jsonDoc to generate documentation)

GET /apidoc.html

Also see the files in org.cresst.sb.irp.rest, which are documented with jsonDoc annotations with a description for each API route.

Data Files

CAT Data Files

The IRP CAT module is designed so that when new updates are made to the data (such as blueprints, item pool, etc.) a plain text file can be edited to update the module. Data files are all stored as CSV (comma separated values).
Requirements:

- Header row will be the first row of the csv file.
- Variable name must appear exactly as they do in the list.
- Order of the fields matters, see the specification for each of the files below.
- Fields are described and the type of the column is put in parentheses. For example, Field 1, (numeric/integer) means that field 1 has a type of integer.

Headers

- **Student Simulated Data [Simulation Package] 3 Columns:**
  - Field 1, unique student identifier (string/alphanumeric) For example: "1"
  - Field 2, unique item identifier (string/alphanumeric) For example: "11785"
  - Field 3, response to item (string) For example: "1" or "0"
- **True Theta Values [Internal file] 2 Columns** (Population parameters used to generate the student responses)
  - Field 1, unique student identifier (string/alphanumeric) For example: "1"
  - Field 2, theta value which represents the population parameters used to generate the simulated responses (numeric/decimal) For example: "-2.66620510066828"
- **Blueprints [Simulation Package] 13 Columns:**
  Each line represents one blueprint statement
  - Field 1, *description* that should be included in the blueprint violation table. For example, "Literary" should be the description for targets 1, 2, 3, 4, 5, 6, 7.
  - Field 2, *subject* (string: [ELA or Math])
  - Field 3, *grade* (numeric/integer)
  - Field 4, *claim* (numeric/integer: [1,2,3,4 for ELA and 1, "4,2", 3 for Math])
  - Field 5, *minimum* number of items needed to satisfy statement (numeric/integer)
  - Field 6, *maximum* number of items needed to satisfy statement (numeric/integer)
  - Field 7, *target* (comma separated list of strings enclosed in double-quotes). Each value in the comma separated string given is tested as an OR logical statement. For example, a value of "1,2,3" tests if the target is either 1, 2 or 3.
  - Field 8, *depth of knowledge* (comma separated list of integers enclosed in double-quotes). Compares if dok is one of the values in the list.

- Field 9, *depth of knowledge greater than or equal*: (dok>=). Compares if dok is greater than or equal to the value given (numeric/integer).
- Field 10, *passage* (ELA only but still need to include blank column for Math) (string/"Long" or "Short")
- Field 11, *short answer* (ELA only, still need to include blank column for Math: (boolean/represented as 1 or 0)
- Field 12, *stim*: (string/"item" or "stim")
- Field 13, *weight* of statement (numeric/decimal)

- **Student Score Data [Vendor Upload] 11 Columns:**
  - Field 1, unique student identifier (string/alphanumeric)
  - Field 2, overall score on logit/theta scale (numeric/decimal)
  - Field 3, standard error of measurement for the overall score on logit/theta scale (numeric/decimal)
  - Field 4, claim 1 score on logit/theta scale (numeric/decimal)
  - Field 5, standard error of measurement for the claim 1 score on logit/theta scale (numeric/decimal)
  - Field 6, claim 2 score on logit/theta scale (numeric/decimal)
  - Field 7, standard error of measurement for the claim 2 score on logit/theta scale (numeric/decimal)
  - Field 8, claim 3 score on logit/theta scale (numeric/decimal)
  - Field 9, standard error of measurement for the claim 3 score on logit/theta scale (numeric/decimal)
  - Field 10, claim 4 score on logit/theta scale (numeric/decimal)
  - Field 11, standard error of measurement for the claim 4 score on logit/theta scale (numeric/decimal)
- **Items Administered [Vendor Upload] 3 Columns:**
  - Field 1, unique student identifier (string/alphanumeric)
  - Field 2, unique item identifier (string/alphanumeric)
  - Field 3, item score (numeric/integer)
- **ELA Item pool [Simulation Package] 34 Columns**:
  - "StimID" stim id (string/alphanumeric)
  - "ItemID" (string/alphanumeric)
  - "Subject" (ELA/Math)
  - "ItemGrade" (numeric/integer)
  - "PoolGrade" (numeric/integer)
  - "Claim" (numeric/integer either 1, 2, 3, or 4)
  - "Target" (string/alphanumeric)
  - "UseTarget" (string/alphanumeric)
  - "BlueprintTarget" (string/alphanumeric)
  - "DOK" (numeric/integer)
  - "AsmtType" (CAT/PT)
  - "ItemType" (string)
  - "IRT_A" (numeric/decimal)
  - "IRT_B" (numeric/decimal)
  - "IRT_C" (numeric/decimal)
  - "IRT_Step1" (numeric/decimal or NA)
  - "IRT_Step2" (numeric/decimal or NA)
  - "IRT_Step3" (numeric/decimal or NA)
  - "IRT_Step4" (numeric/decimal or NA)
  - "IRT_Step5" (numeric/decimal or NA)
  - "IRT_Step6" (numeric/decimal or NA)
  - "IRT_Step7" (numeric/decimal or NA)

    - "IRT_Step8" (numeric/decimal or NA)
    - "Braille" (BRF or empty)
    - "EnemyItem" (ItemID or empty)
    - "ExtPool" (boolean represented as 1 or 0)
    - "ShortAnswer" (boolean represented as 1 or 0)
    - "FullWrite" (boolean represented as 1 or 0)
    - "Passage" ("Short", "Long", or "NA")
    - "WriteRevise" ("ReviseBrief", "WriteBrief", or NA)
    - "Claim2cat" (EE, OP, or NA)
    - "StimMax" (numeric/decimal or NA)
    - "StimMin" (numeric/decimal or NA)
    - "SpaErrors" (numeric/decimal)
- **Math Item Pool [Simulation Package] 26 columns**:
    - "ItemID" (string/alphanumeric)
    - "Subject" (Math or ELA)
    - "ItemGrade" (numeric/integer)
    - "PoolGrade" (numeric/integer)
    - "Claim" (numeric/integer: either 1, 2, 3, or 4)
    - "Target" (string/alphanumeric)
    - "DOK" (numeric/integer)
    - "StimID" (all NA for Math)
    - "AsmtType" (CAT or PT)
    - "ItemType" (eq, gi, mc, mi, ms, sa, ti)
    - "Calculator" (No or Yes)
    - "IRT_A" (numeric/decimal)
    - "IRT_B" (numeric/decimal)
    - "IRT_C" (numeric/decimal)
    - "IRT_Step1" (numeric/decimal or NA)
    - "IRT_Step2" (numeric/decimal or NA)
    - "IRT_Step3" (numeric/decimal or NA)
    - "IRT_Step4" (numeric/decimal or NA)
    - "IRT_Step5" (numeric/decimal or NA)
    - "IRT_Step6" (numeric/decimal or NA)
    - "IRT_Step7" (numeric/decimal or NA)
    - "IRT_Step8" (numeric/decimal or NA)
    - "Braille" (BRF, Not Braillable, PRN, or empty)
    - "Translation" (string: spa or NA)
    - "ExtPool" (boolean represented as 1 or 0)
    - "SpaErrors" (numeric/integer)

Data File Folder Format

- By default IRP expects the CAT Data files to be stored in **/CAT_simulation_packages**
- Format of the data folder is as follows
    - **<SUBJECT><Grade>/**
        - For example **ela3/** contains data files for ela grade 3 (Subject should be all lowercase)
        - Contains 2 necessary files (need to be gzipped)
            - **TrueThetas_ela3N1000ASL.csv.gz**
            - **studentItemResponses_ela3N1000ASL.csv.gz**
    - **ItemPools/**
        - Contains all item pool information

        - Needs 2 files:
            - **ela_itempool.csv**
            - **math_itempool.csv**
    - **Blueprints.csv**

Revision History

Date	Version	Description	Author
2017-02-23	2.0.0	Initial document	Stefan Eng