# IDS Transposer Software Design Document

Ashok Bhargav, University of Michigan
Jane Wang, Ford Motor Company

**Section 1 – Project Description**

1.1 Purpose of this Document

This document contains technical information on the Intermediate Data Systems (IDS). This will be useful to anyone who wishes to get a technical understanding of IDS system. Developers can use this information to enhance the IDS or to develop a standalone version of the application.

This document also explains the basic design algorithm of IDS, and illustrates key components of the application with code snippets.

The information in this paper will enable a developer to understand and configure a run-time environment in which to run this application, or a workspace in which to enhance the software. This document does not exhaustively describe the IDS application; rather it serves as a quick technical guide for a developer trying to install or enhance the software.

1.2 Description

The Intermediate Data Structure (IDS) for Longitudinal Historical Microdata application transforms longitudinal data into a simple data format that is human-readable and can be used as input to relational database applications.

The application has also been written to make it easy for interested developers to enhance the software and deploy it to their local environments. This document does not describe the data mapping in detail, but it will provide references to resources that provide more detail.

The IDS application uses the ICPSR framework for authentication, authorization, and email support. Developers can easily decouple ICPSR-specific utilities to make the software work in their custom framework.

1.3 Requirement

Because the output is sent as an email attachment, the only end user requirements are a working email address and internet access. It is also recommended that end users take steps to ensure that email from icpsr.umich.edu isn't directed into their spam folders.

## 2 Application Design

### 2.1 Technologies used

IDS is a J2EE application hosted among a suite of ICPSR applications. The technologies used to develop IDS are HTML, JavaScript, CSS, Twitter Bootstrap, JAVA, J2EE Application servers, Oracle RDBMS, CVS and GIT repositories. Eclipse is the IDE used to develop this application.

#### 2.1.1 Data Design

The input files are listed below. Please refer to "IDSTransposerUserGuide" for a detailed description of each of these files.

#### 2.1.2 Entity Mapping File

The entity mapping file describes the relationship between the Individual and the Context.

#### 2.1.3 Relationship Mapping File

The relationship mapping file specifies the relationship between Individuals and Individuals.

#### 2.1.4 Data files 1 through 20

The application accepts up to 20 data files to be transposed.

### 2.2 Output File

#### 2.2.1 Transposed output

The application sends out an email with an attachment named "Result.zip" that is composed of Context, Context-Context, Individual, Individual-Context, and Individual-Individual files.

### 2.3 Logging

The IDS application uses Apache Log4j to log events in the IDS Controller. All major events while processing IDS data files are recorded, which will help in debugging the application. In the example below, the creation of the zip file logged.

```
Logger log = Logger.getLogger(IDSTransposerController.class);
log.info("created zip file " + zipFile.getAbsolutePath());
```

### 2.4 Zip Utility

The application uses Java's built-in ZIP utility to compress processed files, bundle them, and email the bundle to users. In the sample method below, a set of output files are zipped together to send to user.

```
private String zipFiles(List<String> fileNames)
```

```
{
        String zipFileName = "";
        byte[] buffer = new byte[1024];

try
{
        File zipFile = File.createTempFile("Results", ".zip");
        zipFileName = zipFile.getAbsolutePath();
        FileOutputStream fos = new FileOutputStream(zipFile.getAbsolutePath());
        ZipOutputStream zos = new ZipOutputStream(fos);
        for (String fn:fileNames)
        {
                ZipEntry ze= new ZipEntry(new
                ZipEntry(fn.substring(fn.lastIndexOf('/')+1)));
            zos.putNextEntry(ze);
            FileInputStream in = new FileInputStream(fn);

            int len;
            while ((len = in.read(buffer)) > 0)
            {
                zos.write(buffer, 0, len);
            }
            in.close();
        }
        zos.closeEntry();
        zos.close();

        log.info("created zip file " + zipFile.getAbsolutePath());

}
catch(IOException ex)
{
        log.info("error when zipping files - " + ex.getMessage());
}

        return zipFileName;
}
```

## 2.5 CSVWriter

The application also uses the Apache CSV utility to read and write comma-separated value files. The method below creates a list of map entries in the input CSV file for further processing.

```
private List<Map<String,String>> readFile(String fileName, boolean validateHeader) throws
IDSException {

        List<Map<String,String>> mapList = new ArrayList<Map<String,String>>();
        try {
                ICsvMapReader csvReader = getReader(fileName);
                String[] header = csvReader.getHeader(true);
                Map<String, String> csvMap;

                while ((csvMap = csvReader.read(header)) != null)
                {
                        Map<String, String> tMap = new TreeMap<String,
                        String>(String.CASE_INSENSITIVE_ORDER);
                        tMap.putAll(csvMap);
                        mapList.add(tMap);
                }

                csvReader.close();
```

```
                if (validateHeader)
                        validateHeader(header);
        }
        catch (IDSException e3) {
                throw e3;
        }
        catch (FileNotFoundException e) {
                throw (new IDSException("File not found - " + fileName + "\n  " +
        e.getMessage()));
        }
        catch (IOException e1) {
                throw (new IDSException(e1.getMessage()));
        }
        catch (Exception e2) {
                throw (new IDSException("The file does not appear to be a valid csv file.
        Please ensure that the first row contains the column names, that each column name
        is unique, and that fields are quoted correctly. " + e2.getMessage()));
        }

        return mapList;
}
```

## 2.6 Data collection

Various Java Collections are used by the application. They include Java File, Lists, Maps, and Arrays. Since the application processes large amounts of data that does not persist in a database, the data is held in memory in these collections.

## 2.7 Software Design

### 2.7.1 Class Diagram

#### 2.7.1.1 IDS Controller

IDS is an MVC (Model View Controller) application. The controller class is responsible for collecting input: entity and relation mapping files, data files 1 through 20, email address, and the file format of the input data. It also responsible for rendering the responses as JSP and emailing

the results to the user after processing is complete.


### 2.7.1.2 Transposer

The Transposer class contains all the business logic for the processing. The Transposer processes the data after validating the header. If the processing is error free, the Transposer creates Excel files for the user.


### 2.7.2 Sequence Diagram

The sequence diagram describes the control flow in the application based on a time sequence. The main components of the application are JSP, IDSController and IDSTransposer. The major methods have been listed below.

- JSP is the presentation layer. JSP renders the web form to capture the data files. When the user submits the web form, the data files and user options are submitted to the IDS controller.

- IDSController :: Validate
  The IDSContoller first validates the data received from the user. If it finds neither mapping files nor data files, an error message is displayed to the user.

- IDSController :: Build IDSTransposer
  Once the data is validated, the IDSController builds an IDSTransposer object. During the build process, the data files uploaded by the user are transferred to IDSTransposer.

- IDSController :: Email Results
  When the IDSController receives the transposed files from the IDSTransposer, the files are then emailed to the user.

- IDSController :: Cleanup
  The IDSController does a cleanup of temporary files used in the process of Transposing.

- IDSTransposer :: Read Entity Mapping
  The IDSTransposer reads the Entity mapping file into memory.

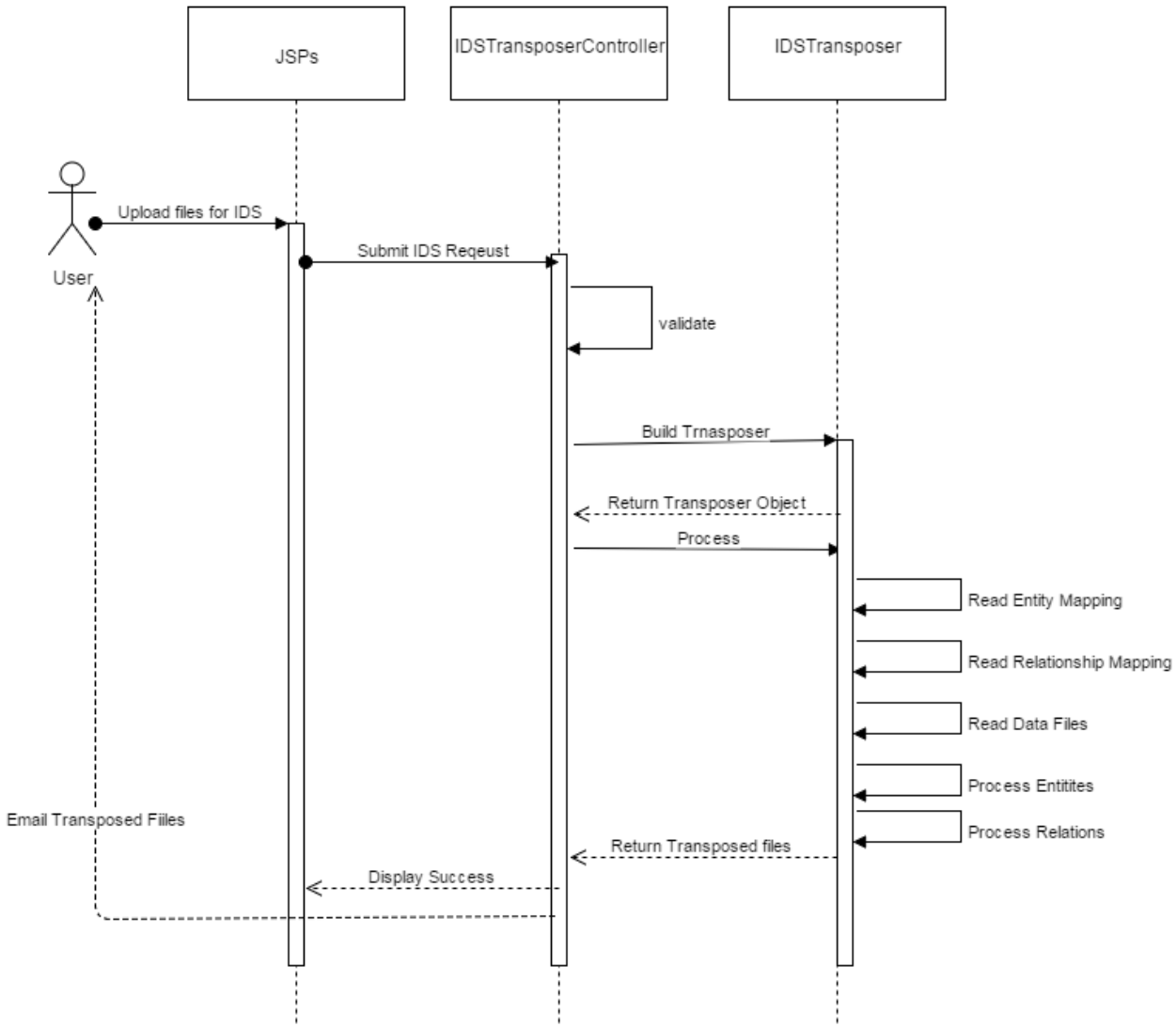
- IDSTransposer :: Read Relationship Mapping
  IDSTransposer reads the Relationship mapping file into memory.

- IDSTransposer :: Read All Data Files (1 through 20)
  The IDSTransposer reads data files into memory.

- IDSTransposer :: Process Entities
  IDSTransposer processes Entities from memory.

- IDSTransposer :: Process Relations
  IDSTransposer processes Relations from memory.

- IDSTransposer :: Build Process
  Build process is initialized.




2.7.3 Limitations

2.7.3.1 File size limit (ISP)

Certain ISPs have limitations of the size of uploaded files. There is a possibility that the file the user trying to upload might not be permitted for upload by his/her ISP.

2.7.3.2 ISP timeout

Depending on the upload speed and file size, the user might face a timeout issue.

2.7.3.3 Email account limit

The user's email account may also have a size limitation. If the user is not getting email back from the application, the user may have exceeded the allotted size of the inbox.

#### 2.7.3.4 Email attachment

There could also be an issue of email clients restricting the size of email attachments. If the user's email provider does not permit the large attachments, the user may need to use a different email provider.

#### 2.7.3.5 Spam filter

The user must also ensure that the email address of the IDS application is not marked as spam. Some email clients automatically flag any address that sends multiple email messages with large attachments and direct emails to a Spam (or Bulk) folder. If the user is not receiving emails from IDS, s/he is advised to check Spam and Bulk fodders.

### 2.7.4 Code

#### 2.7.4.1 Coding Conventions

Code is written to follow industry conventions for Java. All constants are in uppercase. The local and instance variables start in lowercase and are kept in camel case. Single line if statements and loops are avoided and { } are used to designate code blocks. The method names also start in a lower case and are written in camel case.

#### 2.7.4.2 Code Samples

The full code is available for download at GitHub.

The code snippet below is the method used to send email to the end user. An email object is created and output Zip files are attached before sending. The email sent using ICPSR SMTP email server; developers will need to customize the code to utilize their own SMTP server, as the ICPSR server will not work for users outside the University of Michigan network.

```
private void emailResult() {
   Email email = new Email();
   email.setTo(emailAddress);
   email.setFrom(FROM_EMAIL_ADDRESS);
   email.setSubject("IDS Processing Results");
   List<String> attachment = new ArrayList<String>();
   attachment.add(zipFileName);
   email.setMultipartMessageContent("Your results are attached here.  Please let us know
if there is any issue.
         Thanks for using the service.  \n\nICPSR", attachment);
   try {
        email.sendMessageWithAttachment();
      } catch (Exception e)   {
        log.error("Error when sending message. " + e.getMessage());
               }
}
```

In the code below, the sample Individual – Individual relationship is written to the output file. The controller loops through the dataset to get the Date Estimation type for each Individual – Individual relationship. This data is saved in an array using Individual ID as the key.

```
private void writeIndividualIndividual(Map<String, String> relationMapping) throws IDSException {

String relation = getValue(relationMapping, RELATION);
String databaseId = getValue(relationMapping, DATABASE_ID);
String fromEntityIdName = getValue(relationMapping, FROM_ENTITY_ID);
String toEntityIdName = getValue(relationMapping, TO_ENTITY_ID);
String sourceName = getValue(relationMapping, SOURCE);
String dateType = getValue(relationMapping, DATE_TYPE);
String dateEstimation = getValue(relationMapping, DATE_ESTIMATION);
String yearName = getValue(relationMapping, YEAR);
String monthName = getValue(relationMapping, MONTH);
String dayName = getValue(relationMapping, DAY);
String startYearName = getValue(relationMapping, START_YEAR);
String startMonthName = getValue(relationMapping, START_MONTH);
String startDayName = getValue(relationMapping, START_DAY);
String endYearName = getValue(relationMapping, END_YEAR);
String endMonthName = getValue(relationMapping, END_MONTH);
String endDayName = getValue(relationMapping, END_DAY);
String relationVariableName = getValue(relationMapping, RELATION_VARIABLE_NAME);

for (Map<String, String> data:dataList) {
lastIndividualIndividualId ++;

String fromEntityId = getValue(data, fromEntityIdName);
String toEntityId = getValue(data, toEntityIdName);
String source = getValue(data, sourceName);
String relationValue = relation;

String dateEstimationValue = "";
String dateMissingType = getValue(relationMapping, DATE_MISSING_TYPE);
String year = getValue(data, yearName);
String month = getValue(data, monthName);
```

```
                String day = getValue(data, dayName);
                String startYear = getValue(data, startYearName);
                String startMonth = getValue(data, startMonthName);
                String startDay = getValue(data, startDayName);
                String endYear = getValue(data, endYearName);
                String endMonth = getValue(data, endMonthName);
                String endDay = getValue(data, endDayName);

                   dateEstimationValue = getDateEstimationType(data, dateEstimation, year, month, day,
                                 startYear, startMonth, startDay, endYear, endMonth, endDay);
                dateMissingType = getDateMissingType(dateMissingType, year, month, day, startYear,
                        startMonth, startDay, endYear, endMonth, endDay);


                if (isBlank(relation) && (!isBlank(relationVariableName)))
                                relationValue = getValue(data, relationVariableName);


                if (!isBlank(fromEntityId) && !isBlank(toEntityId)) {
                String[] individualIndividualArray = {Integer.toString(lastIndividualIndividualId),
                        databaseId, fromEntityId, toEntityId, source, relationValue, dateType,
                        dateEstimationValue, dateMissingType, year, month, day, startYear, startMonth,
                        startDay, endYear, endMonth, endDay
                    };
                    try {
                        individualIndividualListWriter.write(Arrays.asList(individualIndividualArray));
                   } catch (IOException e) {
                        System.out.println("Error when writing the line - " +
individualIndividualArray.toString());
                        throw (new IDSException("Error when writing the line - " +
                        individualIndividualArray.toString() + e.getMessage()));
                 }
                        }
                }
        }
```

### 2.7.5 Future Enhancements

The application is written on ICPSR web framework. The advantage of a web application is that it can be freely accessed by anyone around the world with little overhead. It does, however, come with the limitations imposed by the web.

- Processing may be interrupted by web server timeout.
- The application is dependent on the speed of your ISP.
- Your ISP may not allow the upload of large files.
- The application cannot simultaneously process more than 20 data files.
- The end user can only run one process at a time, unless s/he opens multiple browser windows to start different sessions.
- Output is only available via email.

Some of these shortcomings can be overcome by modifying the application to work as a standalone application. Most of the business logic will continue to work as-is. As the web application is written in

Java, it is a simple task to set up a Java standalone runtime and development environment. The application can take input from the command line or utilize a simple GUI interface.

- The application can run in a local environment and does not have restrictions on file size.
- Running the application locally will eliminate any timeout concerns
- The number of files processed at a time is not a limiting factor when processed locally.
- The application can be made to process multiple sets of files as a batch process.
- Files can be processed asynchronously without relying on email to deliver processed data.