**Thomas McGee**

**DIS Conversion Tool**

**Design Specifications**

|  |  |  |
| --- | --- | --- |
| **Template Revisions** | **Date** | **Change** |
| V1.0 | 02/07/2012 | Initial Version |
| V1.1 | 02/28/2012 | First Revision |
|  |  |  |



© 2012 CSC. Falls Church, Virginia. All rights reserved. No part of this publication may be reproduced by any means without written permission from CSC. Printed in U.S.A. All questions regarding this documentation should be routed through customer assistance, Blythewood, SC, 800-420-6007 or email [pcssupport@csc.com](mailto:pcssupport@csc.com).

#### Table of Contents

[1. Introduction 4](#_Toc317782863)

[2. Requirement Overview 4](#_Toc317782864)

[2.1. Assumptions 4](#_Toc317782865)

[2.2. Current Scenario 4](#_Toc317782866)

[2.3. Proposed Scenario 5](#_Toc317782867)

[3. Solution 5](#_Toc317782868)

[3.1. DIS Conversion Tool 5](#_Toc317782869)

[3.2. Format Definition File 5](#_Toc317782870)

[3.2.1. Parsing of the Format Definition File 8](#_Toc317782871)

[3.3. In-Memory Table Representation and Mapped Field Types 8](#_Toc317782872)

[3.4. Flat File Processing 9](#_Toc317782873)

[3.5. Access File Load 10](#_Toc317782874)

[4. Description of Screen Form or Page 10](#_Toc317782875)

[4.1. Use Cases 11](#_Toc317782876)

[5. Database Design 15](#_Toc317782877)

[6. Processing Edits 16](#_Toc317782878)

[7. Errors/Error Handling 16](#_Toc317782879)

[8. Interdependencies and Interactions 16](#_Toc317782880)

[9. Interface Information 16](#_Toc317782881)

[10. Application / Technical Impact 16](#_Toc317782882)

[11. Performance 16](#_Toc317782883)

[12. Security 16](#_Toc317782884)

[13. What to Test 17](#_Toc317782885)

[13.1. Test Cases 17](#_Toc317782886)

[14. Specification Signoff 18](#_Toc317782887)

[15. References 19](#_Toc317782888)

[16. Specification Change Log 20](#_Toc317782889)

# Introduction

In order to facilitate Thomas McGee’s (the client) use of the Data Analytics (DA) Data Import System (DIS) Employee import module to import employee information received from one of their vendors, a tool must be created to convert the data received from this vendor into a format acceptable to DIS. This tool will only be used to convert the client’s file from its original format to one that can be accepted by DA DIS, the client will still be required to run the DA DIS process manually.

# Requirement Overview

This file conversion tool will be written in C# using the Microsoft .NET framework 3.5 or higher (4.0 most likely.) This in turn will necessitate the use of Windows XP or higher on client machines. Any processes the tool is to perform will not have any computing power requirements over and above those of the operating system itself so no specific system requirements are necessary. There is a high likelyhood that, in order to facilitate the creation of the DIS-compatible file, any client machine will require software libraries that enable the reading and writing of Microsoft Access 2003 (MDB) or 2007 (ACCDB) files to the extent that these libraries cannot be normally packaged with the final product.

The key features of this application are as follows:

* Parse and interpret a file layout description definition file.
* Import an arbitrary fixed-width file that adheres to the layout definition supplied via the   
  preceeding item.
* Map the defined fields into an internal representation of the DA DIS input Microsoft Access Database file.
* Apply “locally unique’ primary and foreign keys to this internal representation of DIS data.
* Load the internal representation from memory into the already existing DA DIS input MS Access file.
* A User Guide documenting Installation and use. Follow the DA 3.2 User Guide format.
* A simple install package.
* Command line functionality so that loading can automated.

## Assumptions

* No modifications will be required for the DA DIS template itself or the RMX optionset interface.
* The application is DA and RMX version agnostic.
* Format definition files will be generated by hand.
* The application’s target users will be using Microsoft Windows XP SP2 or higher with .NET Framework 4.0 or higher.
* What are the details of the client’s environment?magic!
* The application will accept existing DIS import files, it will not create new versions.

## Current Scenario

There is currently no past or present solution.

## Proposed Scenario

To enable clients with arbitrary, third-party data files they wish to import into RMX via the DA DIS template, this new application would allow them to easily load this file into an Access database file for processing by the DA DIS template.

# Solution

## DIS Conversion Tool

The DIS flat file conversion tool will be a simple application for loading flat files as defined by a format definition file into a DIS Access database file. It will feature a simple interface, a simple workflow and limited or no options and/or interface customization. Primary user functions will consist of specifying files, initializing the import process and optionally clearing or saving log files. The tool should asynchronously keep the user apprised of the import process status and should also provide clear, understandable error messages in cases of abnormal execution or invalid input.

## Format Definition File

The Format Definition File will contain the data required by the application to determine how fields present in the input data file are to be mapped into the output DIS Access file. As this project does not include any kind of automation or machine assistance with generating a FmtDef file, this file will need to be generated manually and must adere to a certain format. The XML Schema for the FmtDef file is as follows. All Format Definitions will be validated against this XSD, formats that do not conform will prevent the execution of a process. CSC will generally be closely involved with any client format definition file generation, but the format is intended to be straightforward enough that an XML novice with appropriate knowledge of the RISKMASTER X database could fashion or modify another format definition after viewing a sample.

<?xml version="1.0" encoding="utf-8"?>

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="FmtDef">

<xs:complexType>

<xs:sequence>

<xs:element name="SkipLines" type="xs:unsignedByte" />

<xs:element name="CommentString" type="xs:string" />

<xs:element name="Tables">

<xs:complexType>

<xs:sequence>

<xs:element maxOccurs="unbounded" name="Table">

<xs:complexType>

<xs:simpleContent>

<xs:extension base="xs:string">

<xs:attribute name="PK" type="xs:string" use="optional" />

</xs:extension>

</xs:simpleContent>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="Fixed">

<xs:complexType>

<xs:sequence>

<xs:element maxOccurs="unbounded" name="Field">

<xs:complexType>

<xs:sequence>

<xs:element name="Start" type="xs:unsignedByte" />

<xs:element name="Length" type="xs:unsignedByte" />

<xs:element name="Table" type="xs:string" />

<xs:element name="Column" type="xs:string" />

<xs:element minOccurs="0" name="Mapping">

<xs:complexType>

<xs:sequence>

<xs:element maxOccurs="unbounded" name="Map">

<xs:complexType>

<xs:attribute name="From" type="xs:string" use="required" />

<xs:attribute name="To" type="xs:string" use="required" />

</xs:complexType>

</xs:element>

</xs:sequence>

<xs:attribute name="Default" type="xs:string" use="required" />

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

An example of a FmtDef file:

<?xml version="1.0" encoding="utf-8"?>

<FmtDef>

<SkipLines>0</SkipLines>

<CommentString>--</CommentString>

<Tables>

<Table PK="ENTITY\_ID">ENTITY</Table>

<Table PK="EMPLOYEE\_EID">EMPLOYEE</Table>

<Table>ICE</Table>

</Tables>

<Fixed>

<Field>

<Start>1</Start>

<Length>8</Length>

<Table>ENTITY</Table>

<Column>BIRTH\_DATE</Column>

</Field>

<Field>

<Start>9</Start>

<Length>1</Length>

<Table>ENTITY</Table>

<Column>SEX\_CODE</Column>

</Field>

<Field>

<Start>9</Start>

<Length>1</Length>

<Table>ENTITY</Table>

<Column>TITLE</Column>

<Mapping Default="">

<Map From="M" To="Mr." />

<Map From="F" To="Mrs." />

</Mapping>

</Field>

<Field>

<Start>65</Start>

<Length>5</Length>

<Table>EMPLOYEE</Table>

<Column>FULL\_TIME\_FLAG</Column>

<Mapping Default="0">

<Map From="FR" To="-1" />

</Mapping>

</Field>

<Field>

<Start>10</Start>

<Length>50</Length>

<Table>ENTITY</Table>

<Column>LAST\_NAME</Column>

</Field>

<Field>

<Start>110</Start>

<Length>12</Length>

<Table>EMPLOYEE</Table>

<Column>SALARY</Column>

</Field>

</Fixed>

</FmtDef>

*FmtDef* serves as the root node.

*SkipLines* allows the application to skip *x* number of lines at the beginning of the file. Optional.

*CommentString* allows the application to skip processing any rows in the file which begin with the specified string. Optional.

*Tables* serves as a meta-element containing a list of the tables required by the mapping, giving the application a heads up about which tables it will need to support in-memory without having to parse through the individual *Target* attributes specified in the following section.

Table defines a table implicated in the format. It uses a single optional attribute, PK to define the table’s primary key, if present. The table name is contained within the body of the element. When a PK is present, the same value will be used for all tables across a row.

*Fixed* serves as a meta-element. in this case indicates that the source file contains fixed-width data. Fixed is declared directly so that a *CSV* layout could be more easily added in the future. Note that support for a CSV layout is currently outside the scope of this document and project. Within the *Fixed* element are the individual field mappings.

*Field* element defines a single piece of data from the source file. Each Field requires four child elements:

* *Start* defines the row character offset where the field begins.
* *Length* defines the number of characters contained in the field.
* *Table* specifies the table the target field resides in.
* *Target* specifies which field in the table in the DIS Access file the data should be loaded into.

Field elements only have to map the fields the user actually wants from the file, the entire contents of a line (consuming the entire length) is not required. Field elements do not need to specify a data type, the data type for a field will be retrieved from the DIS Access file.

To aid in supporting additional types of data files, the specification for a field can also include a simple mapping of known values to other values. This is achieved by including a *Mapping* element in a *Field.* The Mapping must always include a *Default* value attribute and at least a single *Map* value. Each *Map* element must include a before-and-after pair of attributes specifying what each value *From* the file is mapped *To*  in the output. Using this is is possible to have a single field on the input apply to multiple fields in the output.

### Parsing of the Format Definition File

This format definition file was originally envisioned as another type of flatfile which would have to be parsed “by hand” as it were, by the programmer. Upon reflection on the suite of tools available to C# for parsing XML the format was repurposed into XML.

The actual point in the program flow the parsing for the Format Definition file can be determined by the developer; it should have no impact on the function or performance of the application. Obviously however, it must be processed before any work is done with the input file. The format file could be parsed either as soon as a format file is specified by the user or at the onset of file processing.

## In-Memory Table Representation and Mapped Field Types

Using the imported field definitions, the program now must build up an internal representation of the data it will be reading in from the import file. Here is some sample c# pseudocode for extracting the schema of an Access table. It extracts the schema information as a table, it does not actually create an in-memory version of the table. This is primarily for extracting data type information for mapped fields:

OleDbConnection conn =

new OleDbConnection(

"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + \_ IMPORT\_FILE\_PATH + ";");

// retrieving schema for a single table

OleDbCommand cmd = new OleDbCommand(TABLE\_NAME, conn);

cmd.CommandType = CommandType.TableDirect;

conn.Open();

OleDbDataReader reader =

cmd.ExecuteReader(CommandBehavior.SchemaOnly);

DataTable schemaTable = reader.GetSchemaTable();

reader.Close();

conn.Close();

foreach (DataRow r in schemaTable.Rows)

{

foreach (DataColumn c in schemaTable.Columns)

{

c.ColumnName;

c.ColumnSize;

c.DataType;

}

}

Important columns are noted In the nested loops.

Ensuring an error free data import is very important to this processing. In order to prevent potential errors in processing the fields mapped via format definition XML should be cross-checked against the table schemas read in from the Access file. Fields which are present in the mapping but missing in the Access database table shall register as useful error messages in the log window and halt any further processing. Once the process is halted, the user will have the opportunity to correct any errors with their files or specify different files and then restart processing. When a process is halted, all format and schema data collected should be deleted as potential changes to those formats or schema invalidate them in a future context.

As stated above, mapped objects are explicitly cast as the type specified for a field in the Access file. Where a type like ‘text’ (Access’ varchar equivalent, unbounded text is ‘memo’ in Access (I’m not actually sure what memo converts to in a data table, perhaps just a string with no length attribute)) has a field length, that field length should be preserved by truncating data if necessary. Ideally any truncation should be avoided by constructing a format definition file that respects the field lengths of the target Access file.

## Flat File Processing

Once the format definition file and the in-memory tables have been processed and created, the next step of the process is the actual processing of the import data file. A rough outline of how this process should proceed:

Open the data file

If the SkipLines element was present in the format definition, read and discard the number of lines it specified.

Begin a line counter at 0

While there are lines remaining in the file

Read an entire line from the data file

If the *CommentIndicator* element was present in the format definition

Check the first *n* characters of the line against the *CommentIndicator* where *n* is the length of the indicator

If they match, skip this line, otherwise continue.

Increment the line counter by 1

For each table implicated in the format definition

Create a new data row for that table.

If defined, set the *PK* field for the table to the counter value

For each field defined in the format definition for the table

Extract the portion of the line as determined by its *Start* and *Length* values

If the field has a mapping defined

Compare extracted field string against specified From mappings

If a mapping is found

Replace the extracted value with the To mapping

If a mapping was not found

Replace the extracted value with the Default

Try to parse the extracted string as the target datatype (including field length via truncation) of the Access file

If it succeeds load the data into the row

If parsing fails

If the "Halt Processing" option is checked

Display a useful error message in the log window and stop processing the file.

If "Halt Processing" is unchecked

Note in the log window that a field did not parse correctly.

Load NULL into the row for that field

Add the row to the in-memory table

Note in the log window the number of lines processed

It is currently unknown how large the potential files clients may be running through the program. In order to give the user some control over an out-of-control load, especially in the case where Halt Processing is unchecked and a large number of data conversion errors are taking place, the ‘Execute’ button on the interface should change into a ‘Stop Processing’-type button. Enabling this in the application will require some asynchronus input processing as these types of single-minded processed tend to prevent the program’s interface from responding while they run. Perhaps this process can be run via the BackgroundWorker class <http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx>

## Access File Load

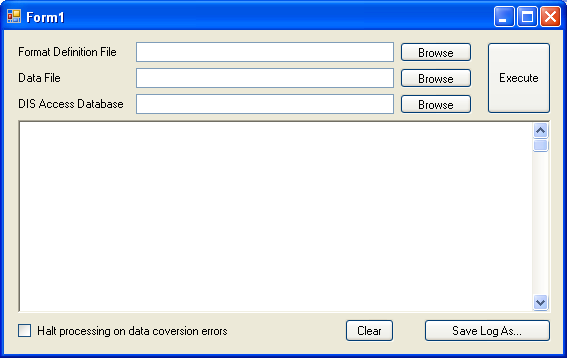
Once the data has been successfully extracted from the file, the contents of the in-memory tables is loaded into the Access file. Since the in-memory tables are based entirely on the Access file itself, there should be relatively few problems at this stage. In the case that problems do occur (most likely as a result of the target access file being open somewhere) this portion of the program should be robustly coded so that all the processed data is not lost in case of error. When an error occurs it should be noted in the error log and the user should be given the option to correct any problems with the Access file and attempt to load the data in again. Allowing for multiple attempts at data load can be conveyed with a renaming of the ‘Execute’ button on the interface.

Since maintaining a reference to the highest identity value used in the supplied Access file adds an unneeded level of complexity to file processing, the Access file load portion will only function when the tables specified in the Format Definition file are empty. The process should proceed normally if tables which are not needed by the format are not empty. Additionally, since the actual Microsoft Access application is only present in higher-end versions of Office, the application should, upon detecting that a table implicated in the format is not empty, give the user the option to clear the contents of those tables (perhaps in the form of a dialog box “Target tables [specify which tables are not empty] are not empty. The import process requires empty tables. Do you wish to empty the listed tables now?”)

Should load into an empty file. Add a routine to clear the access database if it contains records.

Elements specified in the XML must be in the Access database.

# Description of Screen Form or Page



## Use Cases

#### Trigger an import process by passing command line arguments to the application

|  |  |
| --- | --- |
| **Use Case:** | Trigger import via command line. |
| **Context of use:** | This use case describes the steps automate execution of an import process by starting the job with command line arguments |
| **End User Goal:** | Trigger a import process without user intervention. |
| **Primary Actor:** | User or Scheduling Agent configured by user |
| **Precondition:** | Program is not already running (might not matter) |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | Entering the entirety of a command line in a command prompt, shortcut command line or batch process |
| **Main Success Scenario:** | 1. User has a valid Format Definition file. 2. User has a data file which matches the format specified in the Format Definition file. 3. User has a DIS Access database file with all implicated tables containing no data. 4. User enters the command line in the form of [Application name] [Format Definition File path] [Data File path] [Access File path] Into the command console/application symbolic link/batch file. 5. User executes command/runs symbolic link/executes or schedules batch process 6. Process runs, ‘Halt Processing on Data Conversion errors’ is implied on. No user interface windows of screens are shown. 7. Regardless of whether or not the process ran successfully, a log file containing all the messages that would have otherwise been displayed in the log window is saved to the active directory for the application and named with a time/date stamp.lects the ‘Browse” button across from the Format Definition File lable and text box. |
| **Alternate Scenarios:** | 1. Optional |

#### Specify a file target for Format Definintion File

|  |  |
| --- | --- |
| **Use Case:** | Specify a file target for Format Definintion File |
| **Context of use:** | This use case describes the steps to specify a Format Definition File. |
| **End User Goal:** | Supply the application with a valid Format Definintion File. |
| **Primary Actor:** | User |
| **Precondition:** | Program is running and in focus. |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | User wishes to specify a Format Definition File to conduct an import. |
| **Main Success Scenario:** | 1. User generates an appropriate Format Definition file using the XML schema listed above and appropriate knowledge of both the RISKMASTER database and the data file they intend on importing. 2. User selects the ‘Browse” button across from the Format Definition File lable and text box. 3. A Select File dialog comes up. 4. The user selects their Format Definition XML file from their computer. 5. User presses the Enter key or clicks OK to confirm their file choice. 6. (Optional) The program identifies the file as valid XML and reports this to the log window. |
| **Alternate Scenarios:** | 1. User visually confirms that the previous Format Definition File they supplied to the program is still valid for the import they wish to run and still in the specified location |

#### Specify a file target for Data File

|  |  |
| --- | --- |
| **Use Case:** | Specify a file target for Data File |
| **Context of use:** | This use case describes the steps to specify a Data File. |
| **End User Goal:** | Supply the application with a valid Data File. |
| **Primary Actor:** | User |
| **Precondition:** | Program is running and in focus. |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | User wishes to specify a Data File to conduct an import. |
| **Main Success Scenario:** | 1. User has prepared a data file in a fixed-width format containg data they wish to load into a DIS Access database for importing via the DA DIS template. 2. User selects the ‘Browse” button across from the Data File lable and text box. 3. A Select File dialog comes up. 4. The user selects their prepared Data file from their computer. 5. User presses the Enter key or clicks OK to confirm their file choice. |
| **Alternate Scenarios:** | 1. User visually confirms that the previous Data File they supplied to the program is still valid for the import they wish to run and still in the specified location |

#### Specify a file target for DIS Access Database File

|  |  |
| --- | --- |
| **Use Case:** | Specify a file target for DIS Access Database File |
| **Context of use:** | This use case describes the steps to specify a DIS Access Database File. |
| **End User Goal:** | Supply the application with a valid DIS Access Database File. |
| **Primary Actor:** | User |
| **Precondition:** | Program is running and in focus. |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | User wishes to specify a DIS Access Database File to conduct an import. |
| **Main Success Scenario:** | 1. User has a current, empty DIS Access database available on their computer. 2. User selects the ‘Browse” button across from the DIS Access Database File lable and text box. 3. A Select File dialog comes up. 4. The user selects the empty DIS Access Database file from their computer. 5. User presses the Enter key or clicks OK to confirm their file choice. 6. (Optional) The program identifies the file as valid Access Database and reports this to the log window. |
| **Alternate Scenarios:** | 1. User visually confirms that the previous DIS Access Database File they supplied to the program is still valid for the import they wish to run and still in the specified location |

#### Enabling or disabling “Halt Processing on data conversion errors”

|  |  |
| --- | --- |
| **Use Case:** | Enabling or disabling “Break on Error” |
| **Context of use:** | This use case describes the step a user would take to enable or disable the option for the import process to stop if a converted field is not compatible with the data type of the target table column |
| **End User Goal:** | Set the option to their desired behaviour. |
| **Primary Actor:** | User |
| **Precondition:** | Program is running and in focus. |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | User wishes to change the state of the “Halt Processing on data conversion errors” option. |
| **Main Success Scenario:** | 1. The User takes note of the current state of the option and decides they would prefer the alternate behavior. 2. The user clicks the check box to flip it’s state. |
| **Alternate Scenarios:** | NA |

#### Initiate Import

|  |  |
| --- | --- |
| **Use Case:** | Initiate an import |
| **Context of use:** | Once the user has specified their three files, they want to start the import process. |
| **End User Goal:** | To transfer the data from their data file into the DIS Access file for use by DA DIS. |
| **Primary Actor:** | User |
| **Precondition:** | Program is running and in focus.  A file is specified for each of the three adjacent file selections. |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | User wishes to import their data file into the DIS Access file. |
| **Main Success Scenario:** | 1. User has completed the steps of “Specify a file target for Format Definintion File” 2. User has completed the steps of “Specify a file target for Data File” 3. User has completed the steps of “Specify a file target for DIS Access Database File” 4. User has confirmed they are accepting the state of the “Halt Processing on data conversion errors” 5. User selects the ‘Execute’ button to begin the processing. 6. Execute button changes to “Stop Processing”, or something in that vein. 7. Logging window updates periodically with information about the status of the file processing. 8. Process completes, logging window indicates this, button returns to ‘Execute’. 9. User optionally validates the presence of the imported data in the DIS Access file. 10. User can now continue on to import the Access database into RISKMASTER via the DA DIS template. |
| **Alternate Scenarios:** | 1. User Attempts to start the import process without having specified all three require files.   Multiple options:   1. Do not enable ‘Execute’ button until all three files have been specified. 2. Supply the user with a useful message indicating why the process did not start in the log window. |

#### Clearing the log window

|  |  |
| --- | --- |
| **Use Case:** | Clearing the log window |
| **Context of use:** | User is running multiple imports, wants to remove log information from the previous process. |
| **End User Goal:** | Clear the contents of the log window so a future execution will be unemcumbered by a previous execution. |
| **Primary Actor:** | User |
| **Precondition:** | Program is running and in focus.  Something is in the log window to clear. |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | User wishes to clear the contents of the log window. |
| **Main Success Scenario:** | 1. User notices that the log window contains information they no longer need. 2. User clicks the ‘Clear’ button at the bottom of the window. 3. The contents of the log window is removed. |
| **Alternate Scenarios:** | None |

#### Saving the contents of the log window to a file

|  |  |
| --- | --- |
| **Use Case:** | Saving the contents of the log window to a file |
| **Context of use:** | User may have encountered errors and wishes to save off a copy of the log to supply to support.  User may have doubts about the XML mapping or the data file and they wish to send evidence of problems with the file to a colleague or third party. |
| **End User Goal:** | Retain the current contents of the log window into a text file . |
| **Primary Actor:** | User |
| **Precondition:** | Program is running and in focus.  Something is in the log window worth saving |
| **Minimal Guarantees:** | End User Goal as mentioned above is achieved |
| **Success Guarantees:** | End User Goal as mentioned above is achieved |
| **Trigger:** | User wishes to save the contents of the log window into a text file |
| **Main Success Scenario:** | 1. User notices that the log window contains information they want to retain. 2. User clicks the ‘Save log as…” button at the bottom of the window. 3. The user is presented with a ‘Save File’ dialog with a default file extension of .TXT 4. User can select a target directory, enter a file name and save the log into that file. 5. User returns to the application after hitting Enter or selecting the ‘Save’ or ‘Ok’ buttons. |
| **Alternate Scenarios:** | None |

# Database Design

The application does not have any direct connection to any in-place databases. All database manipulation occurs through use of the DIS Access database file. The schema of this database file is determined by the DIS DB Tools application.

The internal scturcture of the tables used by the application is based entirely on those tables present in the DIS Access file. It does not make any schema changes to any of these databases or tables.

# Processing Edits

Character strings read in from the data file rows should be stripped of excess leading and trailing whitespace.

[Numerical types].TryParse may succeed on strings like addresses which begin with a number, consider checking them against a pattern in addition to tryparsing them.

# Errors/Error Handling

Moderately robust exception handling should be in place for any action involving the Access file, data type conversions, file operations and any other place where the possibility of exception is more than nominal (use your judgement.) All captured exceptions should be reported to the logging window along with any appropriate (and present) inner exceptions.

As mentioned before, even when a user has selected not to stop processing the data file when fields fail a conversion, all conversion errors should still be logged to the window so that the user has useful visual feedback on the quality of their file (alternately, their mapping) and can choose to stop the processing if they choose. Conversion errors should display as follows:

“[DATA\_ELEMENT\_STRING]” failed to convert to [DATATYPE] for [TABLE].[FIELD]

Example:

“Smith” failed to convert to System.Float for EMPLOYEE.HOURLY\_WAGE.

# Interdependencies and Interactions

The entirety of this application is separate from any RISKMASTER module or library or data source. No actions should be required for any potential interactions and no interdependencies should exist.

# Interface Information

No Interface Information modifications are required for this application.

# Application / Technical Impact

.

# Performance

No Performance considerations are required for this program. However, if memory usage or processing time become an issue through testing, they could be added at a later time.

# Security

While Access database files do contain the ability to be password protected, DIS does not feature the ability to construct more than a simple connection string (Provider, file name.) This application will do the same. Password-protected Access databases are not supported.

Additionally, it is assumed that all files supplied to the application will be (roughly, fixed-width files ain’t always preety) human-readable, non-encrypted, UTF-8 or equivalent.

# What to Test

Testing Environment:

Access 2000, 2003, 2007 DIS Files

Windows XP+ .Net 4.0+  
 Various file configurations re: mapped fields, total file size.

## Test Cases

This will be an overview of potential test cases, see the test plan for more information.

1. Supplying invalid XML for format definition.
2. Supplying XML that does not match format definition XSD
3. Supplying a valid format definition
4. Supplying empty data file
5. Supplying valid data file
6. Supplying a very large file
7. Supplying valid data file with lines that are too short for the specified format (out of array bounds)
8. Mismatched XML and data file with ‘Halt processing’ checked
9. Mismatched XML and data file with ‘Halt processing’ unchecked
10. Access file missing tables specified in XML
11. Access file missing fields specified in XML
12. Access file matching XML
13. Access file specified already open elsewhere (locked)
14. Stopping a process immediately after starting it
15. Stopping a process partway through processing a file
16. Stopping a process while the data is being loaded into Access file
17. Saving empty log
18. Saving non-empty log
19. Clearing clear log window
20. Clearing non-clear log window
21. Starting via command-line with no arguments
22. Starting via command-line with incomplete arguments
23. Starting via command-line with invalid file paths
24. Starting via command-line with valid xml but mismatched data
25. Starting via command-line with non-empty Access file
26. Starting via command-line with invalid xml
27. Starting via command-line with all valid files.
28. (Optional) Starting via command-line in a read-only directory (loging)

# Specification Signoff

**The signatures below represent approval and signoff of the requirements and approaches as listed above**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Reviewer(s) Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing Lead Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program Manager Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Development Manager Date

Comments: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# References

DA DIS Database Tools Design Document

# Specification Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change # | Requirement # Revised | Nature of Revision | Initials |
| 02/17/2012 | 0 |  | Original Version | BS |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |