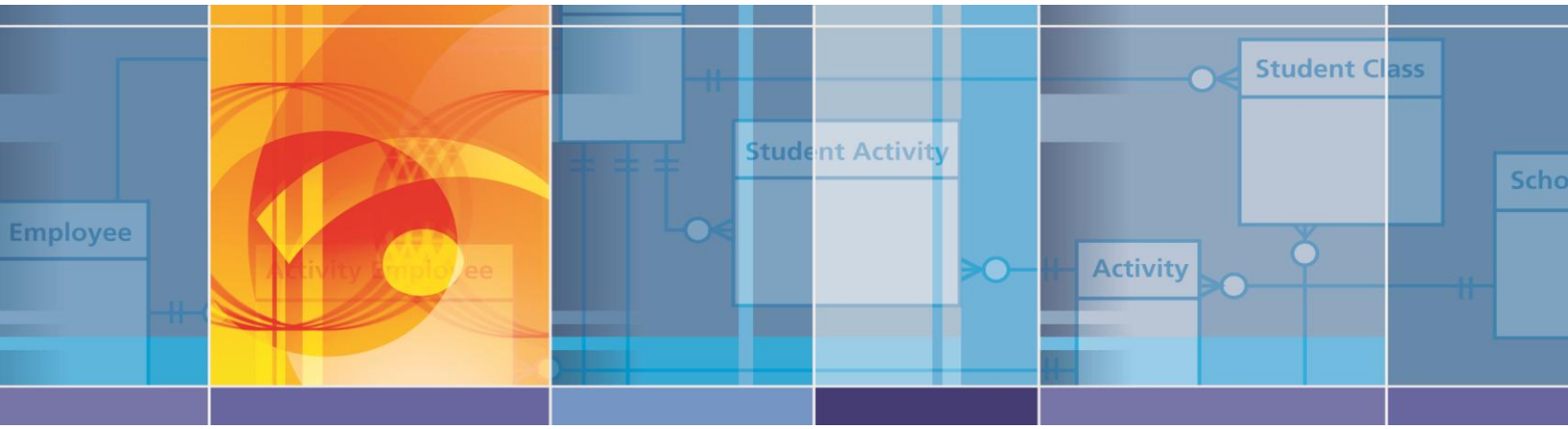


Development Guide

EDFacts Shared State Solution

ESP Solutions Group

Authors: Steven King
Date: 27 June 2014
Version: 0.5

Change History Log

Date	Pages	Summary of Changes	Version	Authorized By
5-Jan-2012		Initial Draft	v0.1	Steven King

Table of Contents

Introduction

EDFacts is the US Education Department's (USED) system for collecting the data required for all USED elementary and secondary education offices and programs. Every state is required to report EDFacts data electronically using the file formats specified by the Department.

Traditionally, a state has a process that goes directly from their source data to the EDFacts files. The EDFacts coordinator put something together that they can use, with little thought about sharing with others – no need—or for building robust monitoring and notification. These state systems often have sketchy or missing documentation.

Many states are in the process of redesigning or rebuilding an EDFacts solution due to the replacement of the current EDFacts coordinator.

The EDFacts Shared State Solution (ES3) is a system states can use to create the files required for EDFacts submission. It uses the standard Microsoft SQL Server stack of applications and standardizes a significant portion of the process.

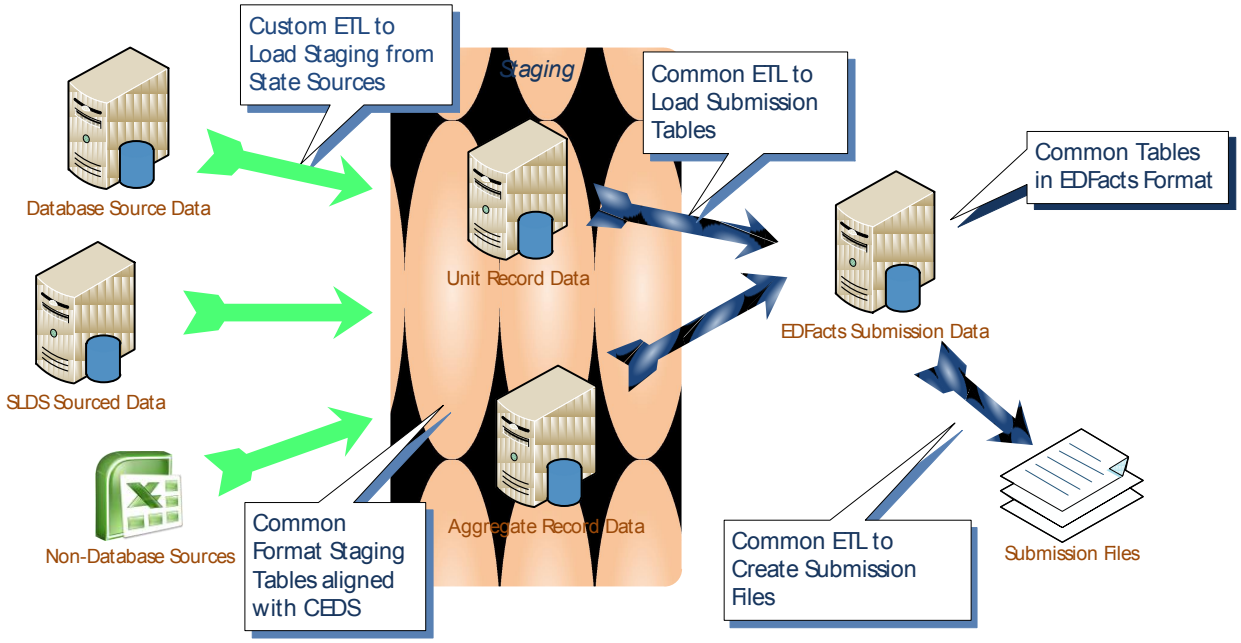
States can get and modify the processes for their custom needs, or engage ESP to do the customization for them.

The ES3 is a solution built with these issues in mind. It is designed to include

- Easy customization and adaptation processes for new states
- Robust design with systems for error catching, monitoring and tracking
- Process logging and monitoring
- Email notification for both success and error processing
- Standard tools and open design so it's easily understood and modified
- Full documentation

System Overview

The ES3 breaks the file creation process into several steps, end with the creation of the EDFacts files.



The submission files, in the bottom left corner of the diagram, have an identical layout for all states.

The ES3 builds a set of tables that mirror the structure of those files. The EDFacts Submission tables have an identical structure for all states. Consequently the ETL from these tables to the Submission Files is common for all.

The ES3 differs from previous solutions by introducing a set of unit or aggregate staging tables (orange section in the middle) that used to construct the EDFacts Submission tables.

To the maximum degree possible, the ES3 attempts to standardize the staging tables across states. Some state may have aggregate source data while others have unit records, but where the detail level is similar, the staging tables have a common structure.

ES3 attempts to align the staging tables with the Common Education Data Standards where that is possible. ES3 does use whatever codes a state may use for a particular element.

By standardizing the staging table design, the ETL process to load the EDFacts Submission tables can be standardized.

Where state customization is expected to occur, information has been extracted out to be managed in simple configuration tables. For example, state codes, state identifier and name, email configuration and staff to be notified, submission file location, etc. are all stored in database tables to be used in the ETL processes.

Customization of the ES3 occurs through the simple editing of database table content for a particular client and reporting period.

The ETL that reads source data and loads the staging tables must be customized for each state. But even here, every effort should be made to standardize where possible and to build templates for each data load.

The staging load ETL processes still include the process logging, monitoring, and notification mechanisms of the Submission Load and File Creation processes.

Objectives

The objective for this document is to provide documentation for how the ES3 is being designed and developed.

The audience is twofold:

1) for the developers to provide guidance for their work and processes to use
2) state EDFacts staff and users to see where and how to modify if required, and instill confidence that the system is being constructed properly.

Scope

The scope is a general description of the processes to be used and the design for the core templates. It is not the details nor documentation for each of the actual ETL processes in the system.

Definitions

Technologies

The EDFacts Shared State Solution is built using a variety of technologies, all of which are in the standard Microsoft stack. The tools include:

SQL	Structured Query Language
SQL Server	Microsoft SQL Server database engine
SSIS	SQL Server Integration Services
SSRS	SQL Server Reporting Services
T-SQL	Microsoft's version of the Structured Query Language procedural programming language
C#	The primary language used for scripting, a .NET language

Relation to the CEDS

The Common Education Data Standards (CEDS) is an effort of the National Center for Education Statistics to develop consistent definitions and uses for education terms.

CEDS does have a logical data model, but that data model is for an operational system; it is highly normalized.

The EDFacts Shared State Solution, both the staging tables and the submission tables, are reporting data bases. They are tuned for the purpose of supporting EDFacts and are "flattened" from a CEDS perspective. The EDFacts Shared State Solution will use the definitions and option sets where ever they make sense.

Design Principles

There are a set of design principles that guide the way the system is developed and deployed.

- EDFacts Shared State Solution is self-contained
- No modifications are required to existing data bases
- The system the file system only to write submission files or invalid record reports
- Core connection information is shared across all packages
- Custom code is encapsulated in stored procedures to the maximum degree practicable
- Users are notified of execution progress via email and SSRS reports
- Standard Microsoft SQL Server tools are used
- Design and build for easy maintenance and modification

Development Plan

Submission Table Load and File Creation

The submission table load and submission file creation process copies data from the staging tables into the submission tables and creates the submission files. The process logs the package start and stop and some of the detail steps. The generic process will validate the staging data and if that passes, loads the submission tables and creates the submission files. At the end of the process, an email is sent to appropriate staff notifying them that the process is complete.

Submission Table Design

For each EDFacts file there is a table in the EDFacts_Submission schema. The EDFacts_Submission tables are designed to mirror the EDFacts formats as much as possible. We will add two columns at the start of the table for the report level and school year (reporting period).

In a few instances, the table structure is different for the different levels, directory for instance. In that case, the level will be appended to the end of the table names: for example S092_SCH or S029_LEA.

If the table structure for the current year is sufficiently different from a prior year for the same submission number, the existing table is renamed by adding the last year for which the table is valid to the end of the table name. Then a new table is created using the naming conventions above. For example, if the 2012-13 membership file, S052, were significantly different than the existing membership file, we would rename the old table to S052_2011_12, and create a new S052 table.

The count field in all the tables is consistently named [totalCount] regardless of the file specification field name. This allows the generic year to year comparison routine to work.

Generic ETL Package Connections

There are 6 connections for the generic Submission File ETL Process. Each is discussed below. A particular ETL package may not have SEA, LEA or School data file connections, if the EDFacts report in question does not include that level.

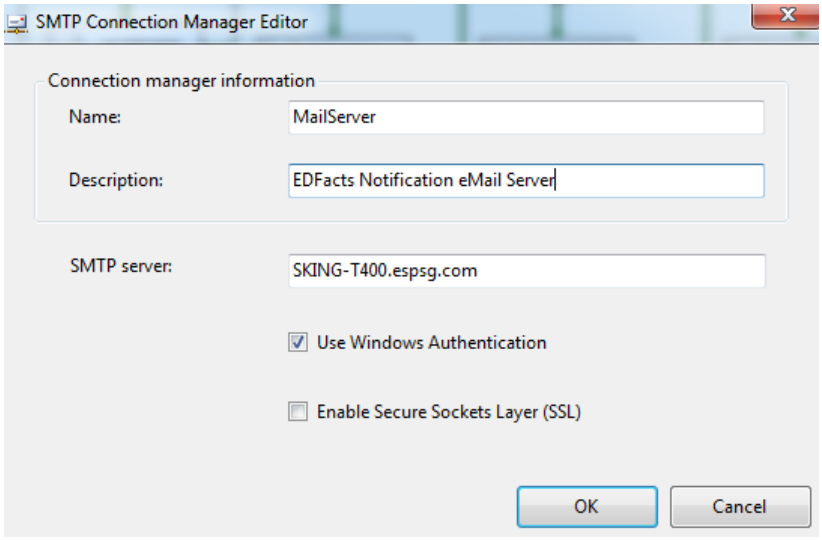
EDFacts

This is an OLEDB connection to the EDFacts SQL Server catalog. This catalog has EDFacts_Admin, EDFacts_staging, and EDFacts_Submission schemas. All three schemas are referenced by tasks in the ETL process.

The connection string for this connection is configured in the SSIS_Configurations table under the Core Connections filter

Mail Server

This is an SMTP connection to the server through which email notifications are sent from the ETL processes.



Change the SMTP Server field to point to the appropriate email server.

The connection string for this connection is configured in the SSIS_Configurations table under the Core Connections filter

Invalid Records

Invalid Records is a flat file connection to where the records are written that fail validation. The file will consist of two columns, one with the key field identifier for the invalid record, and the second for the reason the record failed.

By default, the invalid records are written to a file in the <Root File Directory>/<Work directory>. The file name for the invalid records file comes from the EDFacts_Admin.SubmissionFileCharacteristic table's invalidRecordsFileName field.

SEA Data File

This is a flat file connection to where the SEA EDFacts Submission file will be written. The general behavior is to write these to:

<RootFileDirectory>/<FileSubDirectory>/<FileNameSEA>

The <FileSubDirectory> value comes from EDFacts_Admin.SubmissionFileCharacteristic and the FileNameSEA is built in the ETL Process then written to EDFacts_Admin.SubmissionFileHistory.

Not all EDFacts files specs have an SEA file, so it is not always created.

LEA Data File

This is a flat file connection to where the LEA EDFacts Submission file will be written. The general behavior is to write these to:

<RootFileDirectory>/<FileSubDirectory>/<FileNameLEA>

The <FileSubDirectory> value comes from EDFacts_Admin.SubmissionFileCharacteristic and the FileNameLEA is built in the ETL Process then written to EDFacts_Admin.SubmissionFileHistory.

Not all EDFacts files specs have an LEA file, so it is not always created.

School Data File

This is a flat file connection to where the School level EDFacts Submission file will be written. The general behavior is to write these to:

<RootFileDirectory>/<FileSubDirectory>/<FileNameSchool>

The <FileSubDirectory> value comes from EDFacts_Admin.SubmissionFileCharacteristic and the FileNameSchool is built in the ETL Process then written to EDFacts_Admin.SubmissionFileHistory.

Not all EDFacts files specs have a School level file, so it is not always created.

Generic ETL Package Variables

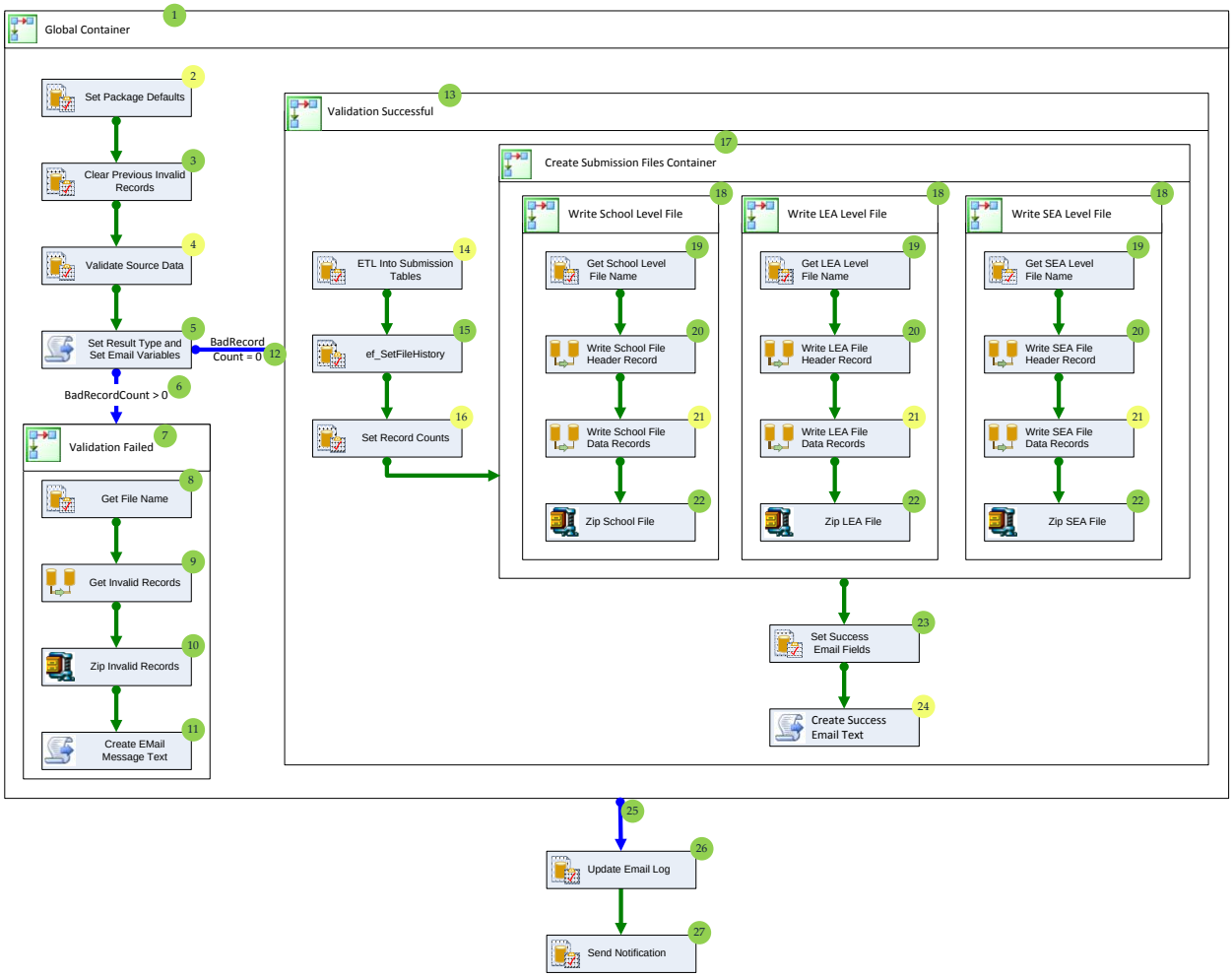
The generic Submission Table load ETL process uses a number of variables in its processing. These variables are shown in the table below. Most of these are implemented as package level user variables in SSIS.

Variable	Value Source	Used By
BadRecordsCount	Originally zero and updated by the [Validate the Source] data flow task	Conditional flows out of the [Log Validation End] task, and the [Validation Failed] [Create Email] task
CsvFileNameLEA		
CsvFileNameSchool		
CsvFileNameSEA		
DescriptivePackageName		
EmailAttachments	Originally blank. Built in the [Validation Failed] or [Validation Succeeded] [Create Email] tasks	[Send Email] and [Update Email Log] tasks
EmailMessageBody	Default values set in the result type task Updated in the [Validation Failed] or [Validation Succeeded] [Create Email] tasks	generate error email message body, generate success email message body, send notification
EmailMessageCCLine	set result type task, read from the EDFacts_Admin.ETLProcess table	send notification
EmailMessageSubject	set result type task, read from the EDFacts_Admin.ETLProcess table	send notification

Variable	Value Source	Used By
EmailMessageToLine	set result type task, read from the EDFacts_Admin.ETLProcess table	send notification
FileSubDirectory		
InvalidRecordsFileName		
PackageLogRowCountSuccesses		
ReportingPeriod		
RootFilePath	read in from the state config table in the Set Filepath routine	send notification
SourcePath		
SubmissionFileNumber		
WorkingDirectory		
ZipFileNameLEA		
ZipFileNameSchool		
ZipFileNameSEA		

Generic ETL Package Flow Design

The generic SSIS package flow is shown in the diagram below.



Each of the numbered items in the flow is discussed below.

The steps with green circled numbers do not require any modification from the template for a specific EDFacts submission file. Only the yellow numbered tasks need to be modified for a specific EDFacts Submission file, specifically:

- Setting the package defaults (#2)
- Validating the Staging Data (#4)
- The ETL of the data from Staging into the Submission tables (#14)
- Set Record Counts (#16)
- Writing the Data Records (#21)
- Creating the Success Email (only if not all three levels are reported) (#24)

All the other steps in the process flow are identical with the template and for each submission file.

1. Global Container

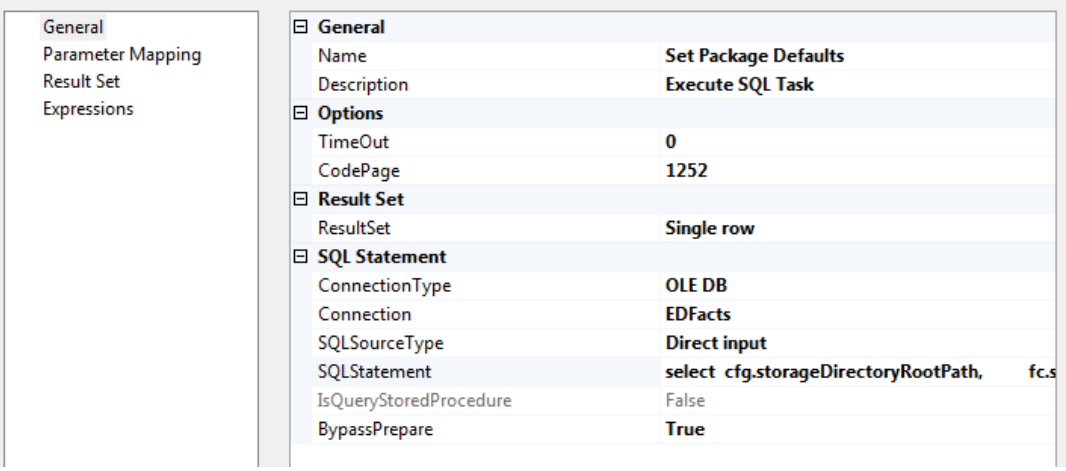
The bulk of the work in the package is contained with a Sequence Container. We do this so that if at any point the SSIS process fails, it will fail over to the final tasks of updating the email log and sending notification. Without this container, the user would never get notified in the event of a failure in the package

No edits are required from the template.

2. Set Package Defaults

This is a SQL Task that reads various values from the configuration tables and saves them into user variables. Values to be set include:

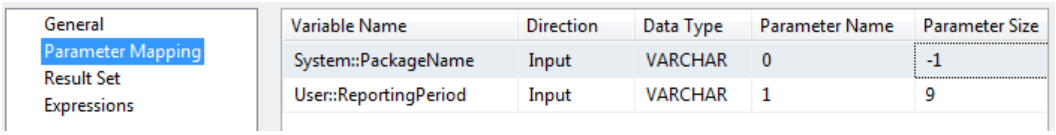
- Root file path
- File Sub directory
- Descriptive Package Name
- Email Message To: Line
- Email Message CC: Line
- Invalid Records File Name
- Email Message From: line



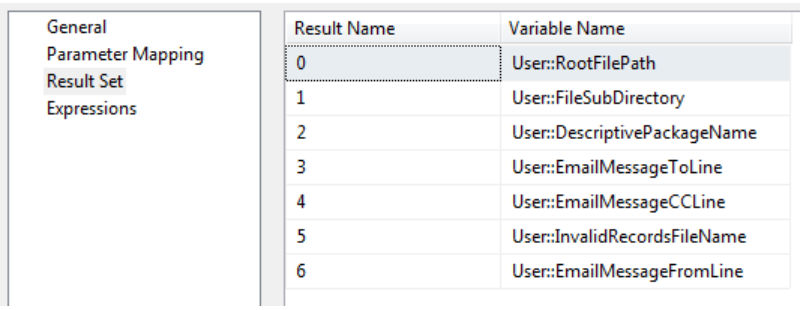
The only customization required for this step for a specific submission file, it to set the specification number in the query as highlighted below.

```
select  cfg.storageDirectoryRootPath,
           fc.fileSubdirectory,
           n.packageDescriptiveName,
           n.errorNoticationToLine,
           n.errorNotificationCCLine,
           fc.invalidRecordsFileName,
           cfg.emailMessageFromLine
   from    EDFacts_Admin.SubmissionFileCharacteristic fc
           join EDFacts_Admin.StateConfig cfg
             on fc.reportingPeriod = cfg.reportingPeriod
           join EDFacts_Admin.ETLNotification n
             on     cfg.reportingPeriod = n.reportingPeriod
                and n.packageName = ?
   where      fc.specificationNumber = 'S052'  ← update with the correct spec number
           and fc.reportLevel = 'SCH'
           and fc.reportingPeriod = ?
```

The only parameter for this step is the reporting period. The Package Name property must be set the same as the package name in the EDFacts_Admin.SubmissionFileCharacteristic table.

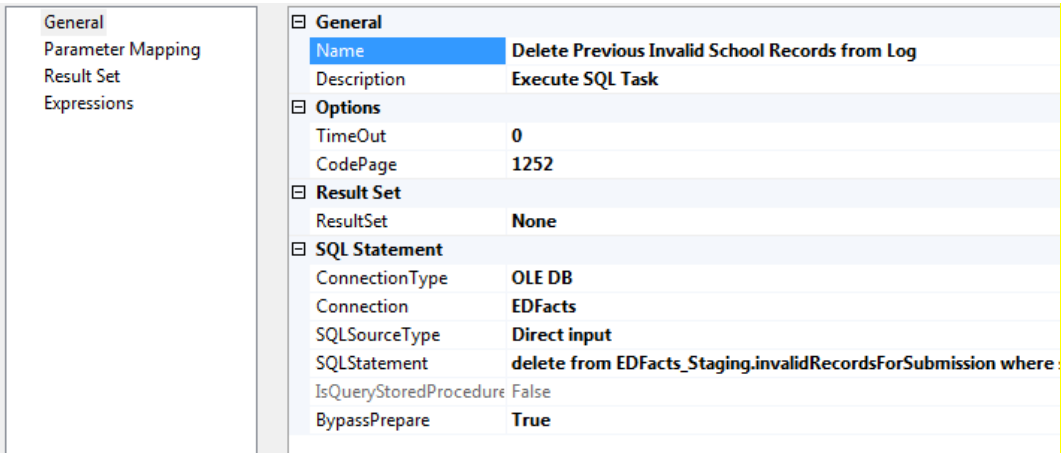


The query reads several fields which then get mapped into the following package level user variables.



3. Clear Previous Invalid Records from the Log

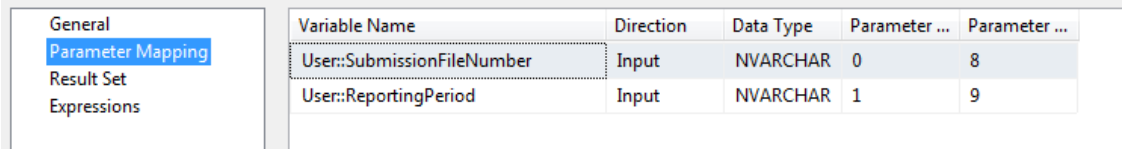
The next task is to clear out any invalid records for this submission file and reporting period from the error log table. If this process has been executed previously there may be records in that table. We will be sending the bad records to the coordinator and don't need to include records that have previously reported. There is not a need to keep a history of the bad records.



This Execute SQL task is a call to a simple DELETE FROM SQL statement

```
delete from EDFacts_Staging.invalidRecordsForSubmission
 where submissionFile = ?
   and reportingPeriod = ?
```

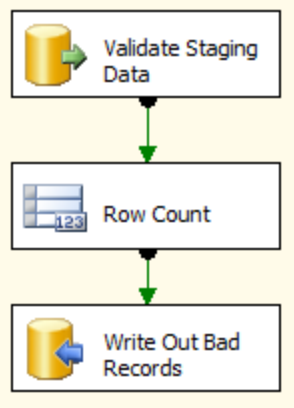
The two parameters for this query are the SubmissionFileNumber and ReportingPeriod.



No edits are required from the template.

4. Validate the Source Data

This is a stored procedure that is custom for each file type and data source.



This data flow task has three steps: query the source for bad records, get a row count of bad records and store that, and finally write the bad records into the bad records table (this is the table cleared out in step 4).

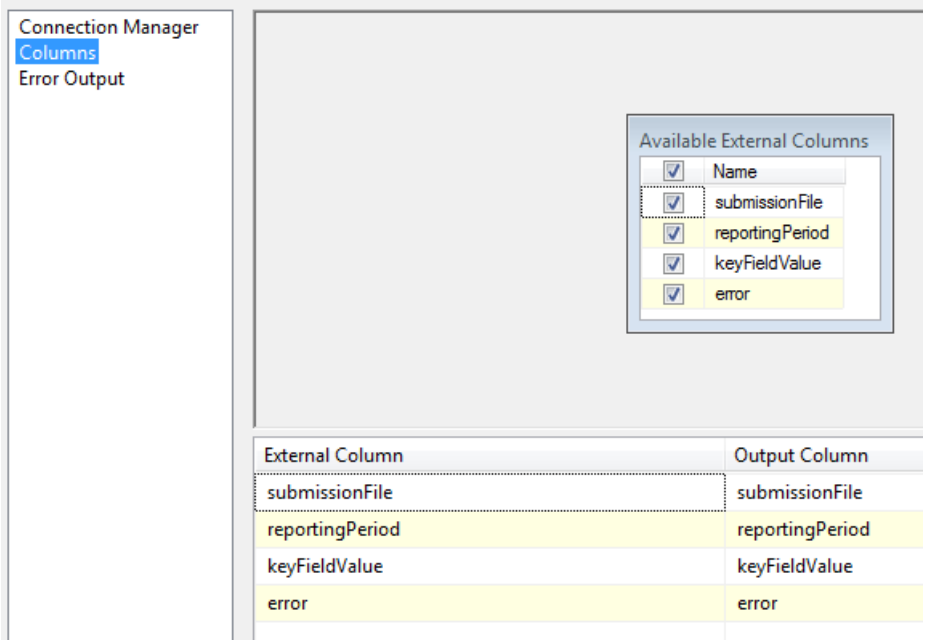
The first step calls a validation stored procedure, specific to the task at hand. In the case of S052, the Membership file, the validation routine might look like:

```
create procedure [EDFacts_Staging].[ef_StageValidation_S052]
  @SchoolYear as varchar (9)
as
  /**
   * Runs validation against the S052 source (Unit_studentDemographics) and
   * SELECTS any bad records.  In the SSIS package, if record count > 0 redirects
   * the process to report the error records and stop the EDFacts file creation
   *
   *  @author      : Steven King, ESP Solutions Group
   *  @version     : 1.0  15-Dec-2011
   *  @param       : @SchoolYear  The school year for which data should be
   *               :              processed in YYYY-XXXX format, for
   *               :              example: 2009-2010
   *  @system      : EDFacts Shared State Solution
   *  Notes        : 'SELECTS' information on the records that fail the validation
   *               : For later insertion into the bad records table. The SELECT
   *               : Statement should generate four fields:
   *               :    FIELD NAME         DATATYPE    EXAMPLE
   *               :    submissionFile     varchar(4)   S052
   *               :    reportingPeriod    varchar(9)   2011-2012
   *               :    keyFieldValue      varchar(50)  987654321
   *               :    error              varchar(250) Race code violation: XY
   *               :
   * Revision History
   * -----------  ---------------  ------------------------------------------------
   * 15-Dec-2011   Steve King       Original draft system built
   */

  begin
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    set  nocount on;

       select   'S052' as submissionFile,
                @SchoolYear as reportingPeriod,
                stateStudentId,
                'Grade Code Violation: ' + gradeLevel as Reason
       from     EDFacts_Staging.Unit_StudentDemographics
       where        schoolYear = @SchoolYear
                and gradeLevel not in
                     ( select  stateCode
                         from    EDFacts_Staging.StateCodeTranslation
```

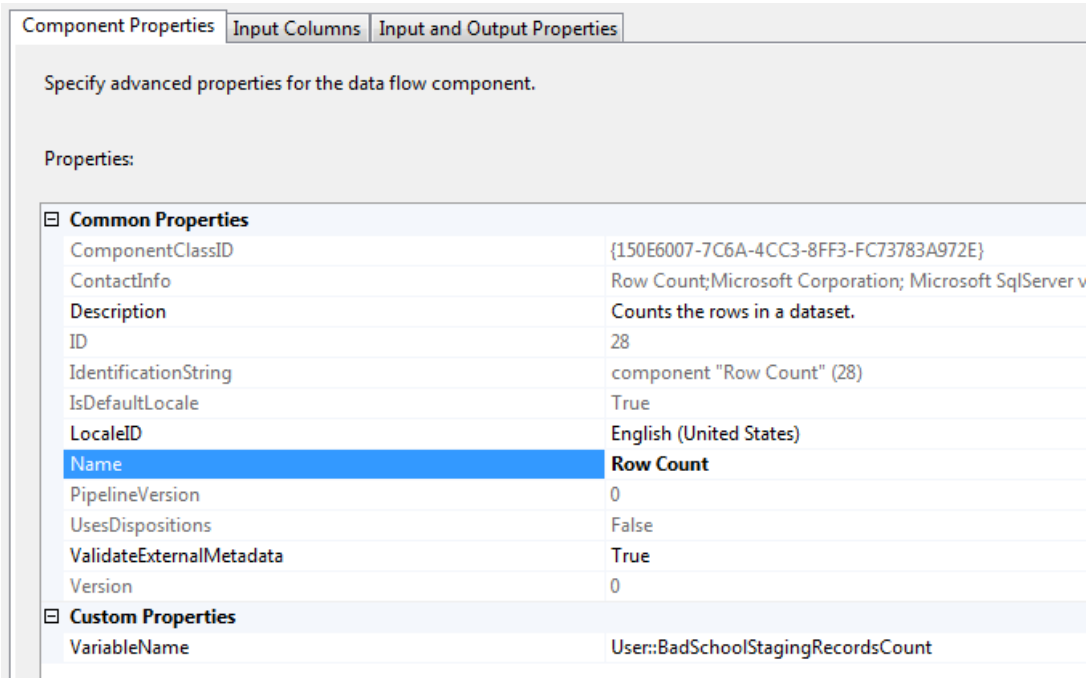
```
                    where   codeSetName = 'Grade Level')
    union
       select   'S052' as submissionFile,
                @SchoolYear as reportingPeriod,
                stateStudentId,
                'Race Ethnicity Code Violation: ' + raceEthnicV2 as Reason
       from     EDFacts_Staging.Unit_StudentDemographics
       where        schoolYear = @SchoolYear
                and left (raceEthnicV2, 25) not in
                      (   select  stateCode
                          from    EDFacts_Staging.
                                    StateCodeTranslation
                          where   codeSetName = 'RaceEthnicity V2')
    union
       select   'S052' as submissionFile,
                @SchoolYear as reportingPeriod,
                stateStudentId,
                'Gender Code Violation: ' + gender as Reason
       from     EDFacts_Staging.Unit_StudentDemographics
       where        schoolYear = @SchoolYear
                and gender not in
                     ( select  stateCode
                         from    EDFacts_Staging.StateCodeTranslation
                         where   codeSetName = 'Gender')
    union
       select   'S052' as submissionFile,
                @SchoolYear as reportingPeriod,
                stateStudentId,
                'School Grade Code Violation: ' + stateSchoolId + '-' + gradeLevel
                  as Reason
       from     EDFacts_Staging.Unit_StudentDemographics
       where        schoolYear = @SchoolYear
                and (rtrim (stateSchoolId) + '-' + gradeLevel) not in
                      ( select  rtrim (stateSchoolId) + '-' + gradesOffered
                          from    EDFacts_Submission.S039
                          where   reportLevel = 'SCH' and schoolYear = @SchoolYear)
end
```

This query has to result in four fields:

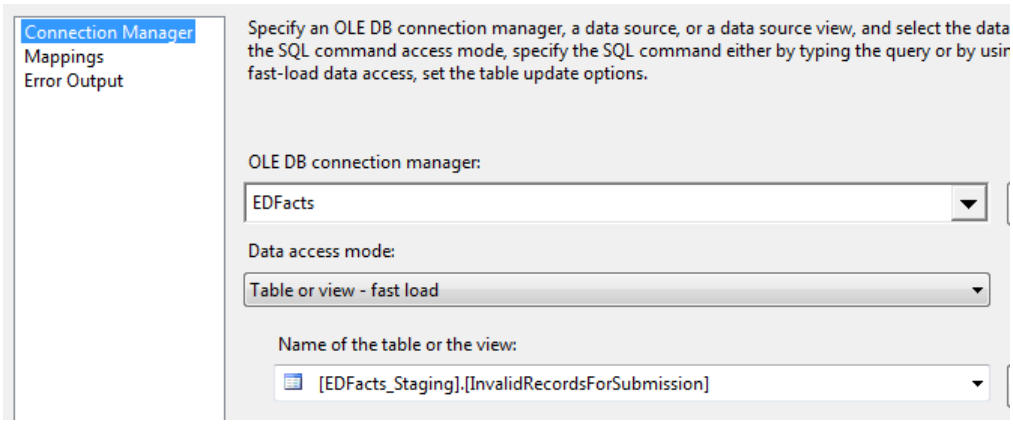
Submission File	The number for the submission file whose staging data is being validated
Reporting Period	The reporting period being covered
Key Field Value	The record identifier for the staging record that violates the validation rule
Error Message	a descriptive message about what validation error was found and used to exclude this record from the Submission file creation process

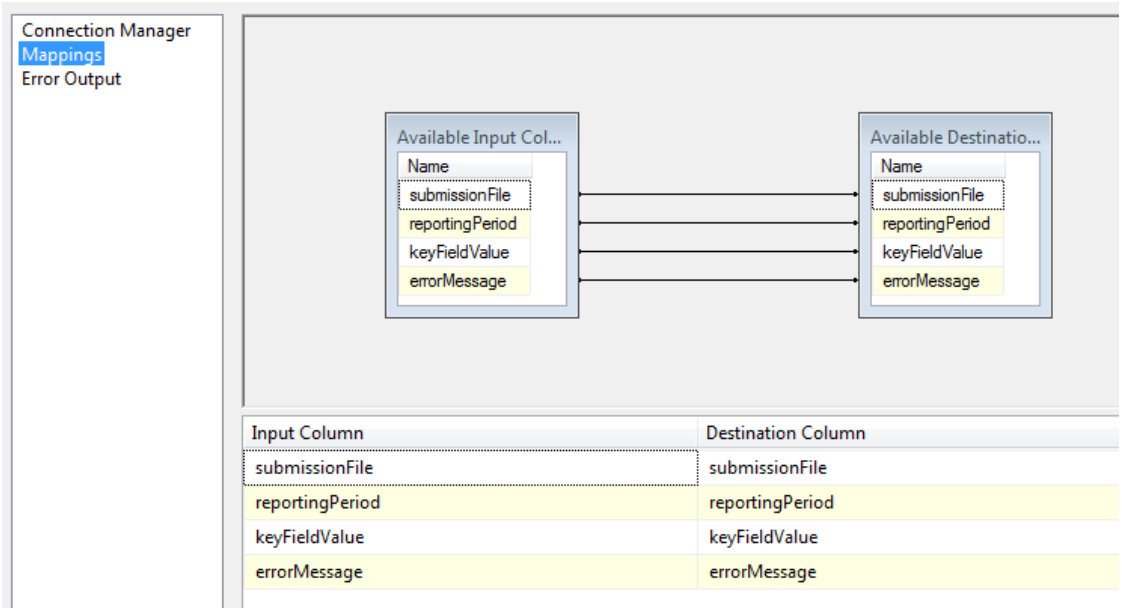


The second step in the validation data flow task is to count the records returned in step 1 and save that count in the variable BadRecordsCount.

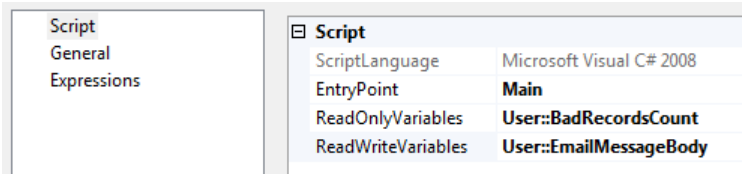


The final step is to save the bad records to the bad records table: EDFacts_Staging.InvalidRecordsForSubmission.





5. Set Result Status and Set Email Variables




```
public void Main()
        {
            //Count processing
            int Failures = (int)Dts.Variables["User::BadRecordsCount"].Value;

            //If failures, set warning
            if (Failures != 0)
            {
                //Get warning message for log
                Dts.Variables["User::EmailMessageBody"].Value
                   = "Warning: "
                     + Failures.ToString()
                     + " validation failures occured";
            }
```

```
            else
            {
                //Get success message for log
                Dts.Variables["User::EmailMessageBody"].Value
                   = "Success: 0 validation errors occured";
            }


            Dts.TaskResult = (int)ScriptResults.Success;
        }
```

No edits are required from the template.

6. Validation Failed Decision

There are two paths out of the validation step depending on if any records failed validation.

This failure path uses an expression constraint that checks if the Bad Record Count (set in step 4) is greater than 0.

No edits are required from the template.

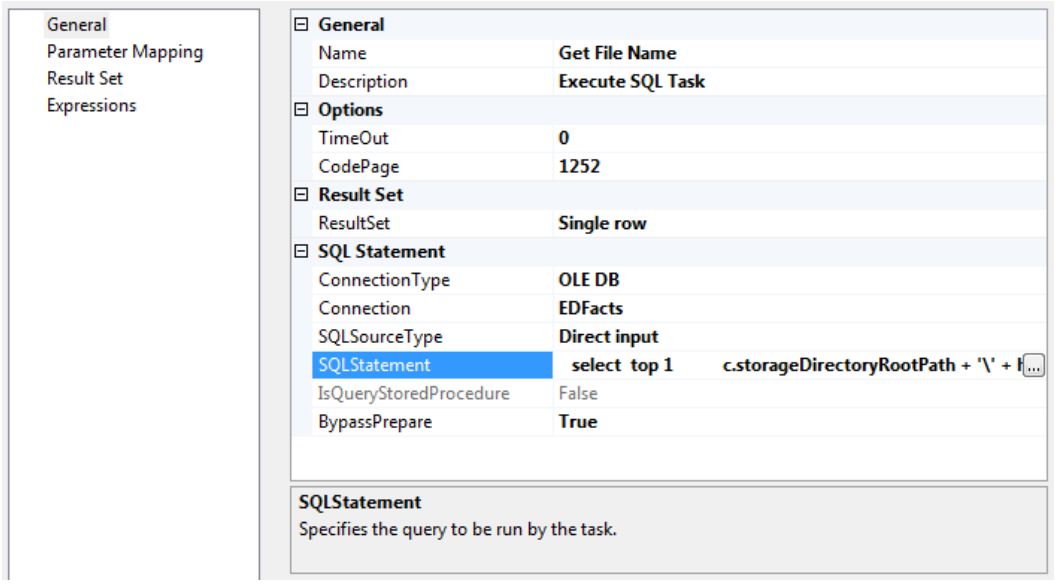
7. Validation Failed Container

This is simply an organizing container for the steps required when the validation process fails.

No edits are required from the template.

8. Get Invalid Records File Name

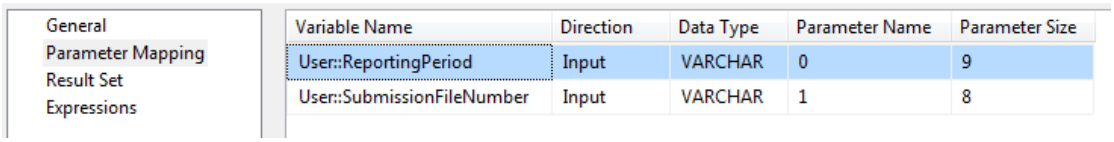
This is a task to get the file names for both the tab delimited and zip file that will store Invalid records. The task also gets the zip file password if one has been assigned.



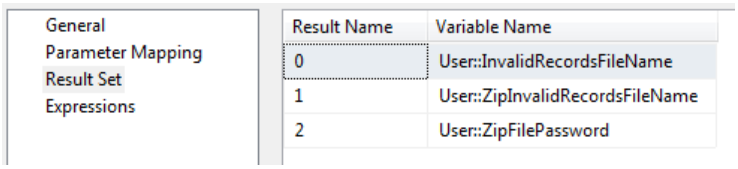
The query that is used in this task is as follows:

```
select  top 1
          c.storageDirectoryRootPath
        + '\'
        + h.fileSubdirectory
        + '\'
        + h.invalidRecordsFileName
        + '.tab'
          as csvFileName,
          c.storageDirectoryRootPath
        + '\'
        + h.fileSubdirectory
        + '\'
        + h.invalidRecordsFileName
        + '.zip'
          as zipFileName,
        h.zipFilePassword
from    EDFacts_Admin.SubmissionFileCharacteristic h,
        EDFacts_Admin.StateConfig c
where       c.reportingPeriod = ?
        and h.specificationNumber = ?
        and h.reportingPeriod = c.reportingPeriod
        and h.reportLevel = 'SCH'
```

This query takes two parameters, the reporting period and the specification number.



And it saves its information into three variables

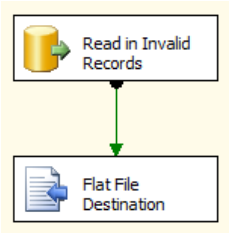


No changes are need for this task from the tamplate.

9. Get Invalid Records

This data flow task reads the bad records out of the bad records table and writes them to the tab delimited flat file. This file will be attached to the error notification email.

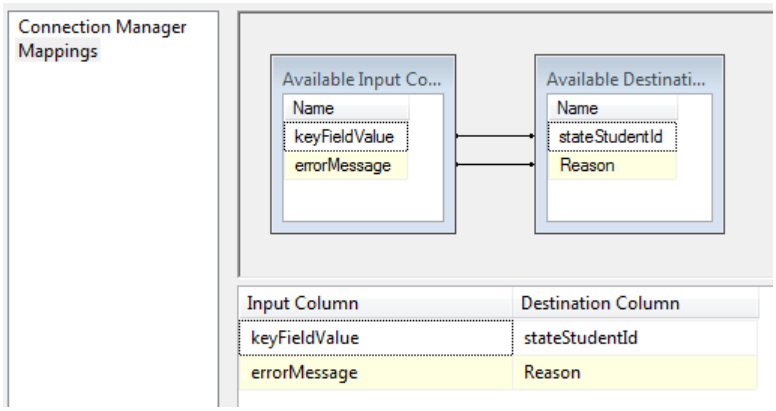
The flow consists of two steps



The first takes the current submission file and reporting period to query out the keyRecordValue and ErrorMsg from the Invalid Records table. The query for the Read step is as follows.

```
select  keyFieldValue,
        errorMessage
  from  EDFacts_Staging.InvalidRecordsForSubmission
  where submissionFile = ?
    and keyFieldValuereportingPeriod = ?
```

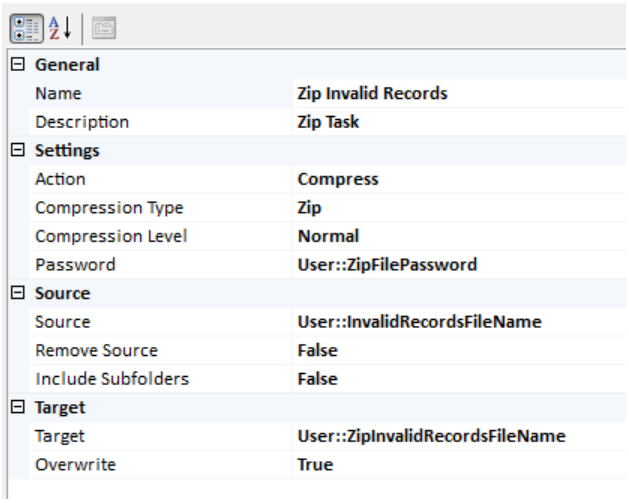
The second step writes these the invalid records flat file.



No edits are required from the template.

10. Zip Invalid Records File

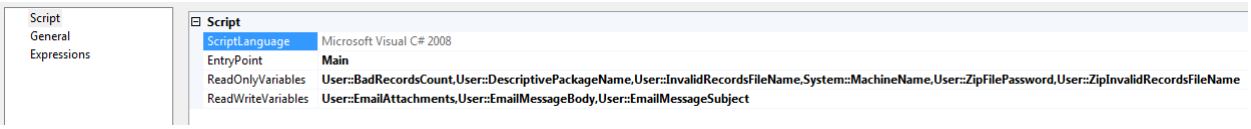
The next task Zips the flat file populated above. The process will apply the Zip Password if one has been defined. The resulting file is written to the ZipInvalidRecordsFileName location.



No changes are needed from the template.

11. Create Error Email Message Body

This script task creates the text for the email notification that will go out and stores the result in the EmailMessageBody user variable. The name of the Invalid Records file is stored in the emailAttachments variable.



The email text states that the validation failed, includes up to the first 20 failing records, and references where the full invalid records list can be found.

```
//Create EMail Validation Failed

public void Main()
  {
    string strMsgText;
    int intBadRecordCount;
    string strWorkFile;
    string strDescriptiveName;
    string strZipFilename;
    string strAttachments;

    strDescriptiveName = (string)
Dts.Variables["User::DescriptivePackageName"].Value;
    strZipFilename = (string)
Dts.Variables["User::ZipInvalidRecordsFileName"].Value;
    strWorkFile = (string) Dts.Variables["User::InvalidRecordsFileName"].Value;

    Dts.Variables["User::EmailMessageSubject"].Value =
       strDescriptiveName
       + " validation failed";
    strMsgText = (string) Dts.Variables["User::EmailMessageBody"].Value;
    strMsgText += "The " + strDescriptiveName;
    strMsgText += " data validation failed on "
       + Dts.Variables["System::MachineName"].Value
       + ".  There ";
    intBadRecordCount = (int) Dts.Variables["User::BadRecordsCount"].Value;

    if (intBadRecordCount != 1)
       strMsgText += "were " + intBadRecordCount.ToString() + " records";
    else
       strMsgText += "was 1 record";
    strMsgText += " that failed.\n";

    if (intBadRecordCount < 20)
       strMsgText += "\nThe record ID's and reasons for the failure are listed
below:\n\n";
    else
       strMsgText += "\nThe first 20 invalid records are listed below: \n\n";


    int i = 0;
    try
     {
       using (StreamReader errRecs = new StreamReader(strWorkFile))
        {
           String line;
           while ((line = errRecs.ReadLine()) != null && i <= 20)
           {
               strMsgText += line + "\n";
           i += 1;
           }
        }
     }
```

```
    catch (Exception e)
    { strMsgText += "          <<<<  Could Not Read the Invalid Records file  >>>>
\n";
       strMsgText += e.Message;
            }

    strMsgText += "\n\nThe full list of invalid records is attached and are
included";
    strMsgText += "in the file: \n    ";
    strMsgText += strZipFilename;
    strMsgText += "\nOn the database server";

    if ((string)Dts.Variables["User::ZipFilePassword"].Value != "" )
    {
       strMsgText += "\n\nThe Zip file has been password protected.";
    }

    Dts.Variables["User::EmailMessageBody"].Value = (object)strMsgText;
    if (intBadRecordCount > 0)
       strAttachments = strZipFilename;
    else strAttachments = "";

    Dts.Variables["User::EmailAttachments"].Value = (object)strAttachments;

    Dts.TaskResult = (int)ScriptResults.Success;
}
```

No edits are required from the template.

12. Validation Success Decision

This step is the alternate path (from step 6) exiting the validation process. It uses an expression constraint checking if the Bad Record Count == 0

No edits are required from the template.

13. Validation Success Container

This is an organizing container for all the steps that occur once the staging data have been validated.

No edits are required from the template.

14. Conduct the ETL

Each EDFacts file type has a matching EDFacts submission table that is loaded by a stored procedure. For most of the EDFacts submissions, the process consists of:

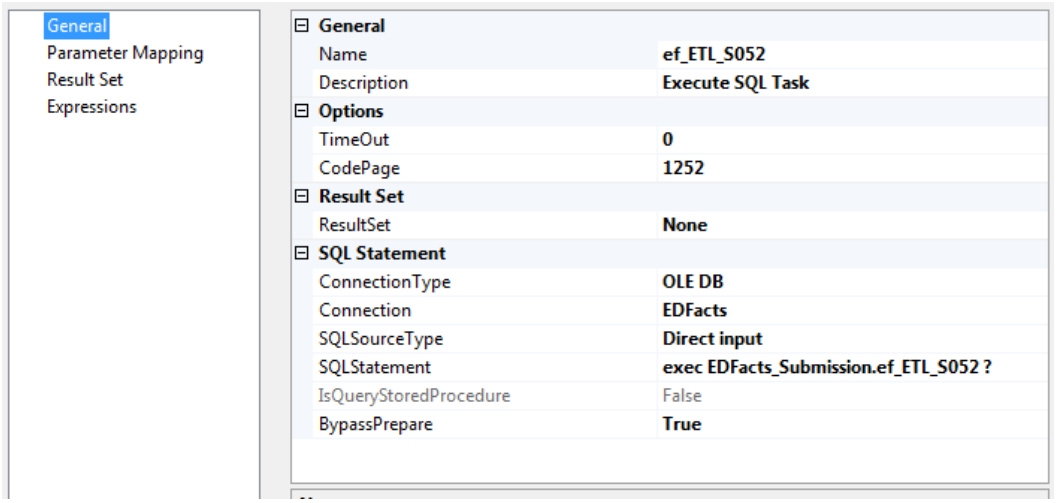
1. Delete any existing records in the submission tables for the selected reporting period.
2. Extract and Load the EDFacts Detail records for the school level. (These will be aggregates of the staging unit records, but the lowest level of detail EDFacts collects)
3. Create the zero records for the detail: create records for any missing subgroups in the data set. For example, a school may not have any 3rd graders for the specific count in question – we need to add a zero record.

4. Calculate the school level subgroups – these are subtotals, but with less detail than the detail level.
5. Extract and load the LEA detail records.
6. Create the zero detail records for the LEA files
7. Calculate the roll-up subtotals for the LEA files
8. Extract and load the SEA detail records
9. Create any zero records for the SEA details
10. Create the SEA level subtotals

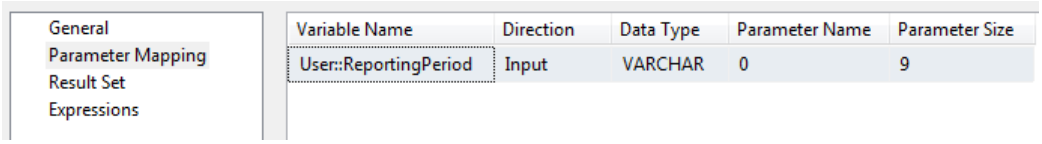
In general the ETL step is an Execute SQL Task that calls a stored procedure for this work. The specific stored procedure should be named like "EDFacts_Submission.ef_ETL_Sxxx" where "xxx" is the submission number, for example "ef_ETL_S052" for the Membership file.

The edits for this task consist of:

1. Making any changes that are required for the stored procedure
2. Making sure the task points to the correct stored procedure

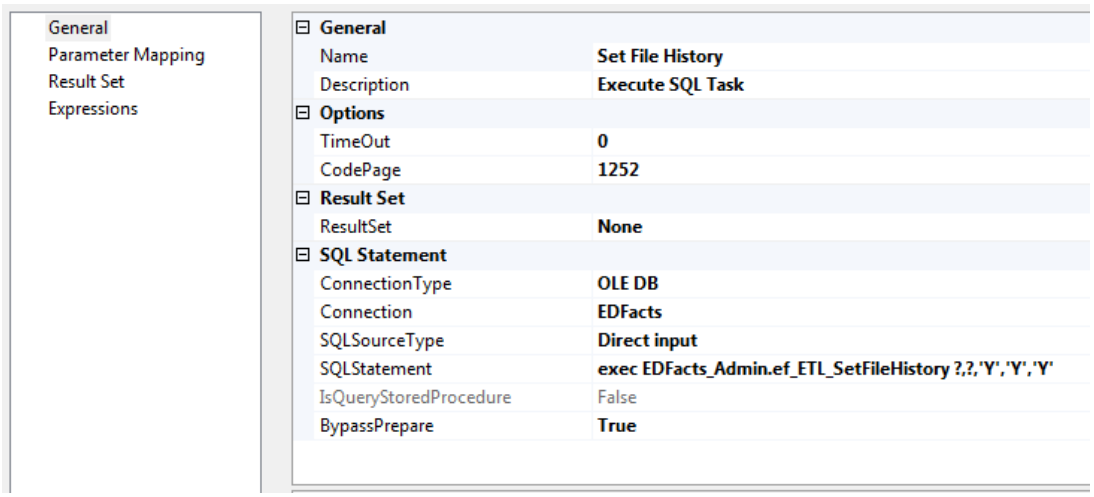


In this case there is a single parameter holding the reporting year.

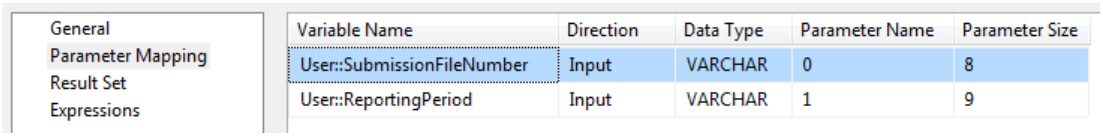


15. Set File History

This is a SQL task that executes the stored procedure ef_ETL_SetFileHistory. The procedure takes five parameters: the submission file number, the reporting period, and then Y/N flags for whether we are creating an SEA file, an LEA file, or a School level file.



We pass in the first two parameters and hard code the last three. By default the last three flags are set to 'Y' in the template.



This stored procedure logs that we are creating files for the selected levels and otherwise populates the EDFacts_Admin.SubmissionFileHistory table. In the process it builds the file name and file identifier that will be used in the headers, and builds the fully qualified name for the tab and zip files that will be created.

No edits are required from the template unless not all three levels will be built, in which case, only the appropriate flag needs to be changed on the SQL Statement line on the General properties page.

16. Set Record Counts

The SET Record Counts task gets the number of records that are going to be written to each of the file levels and updates the SubmissionFileHistory table with that information.

The code that does the record count is shown below.

```
declare @ReportingPeriod   varchar (9) = ?;

   /* School Level Record Count */
   update  EDFacts_Admin.SubmissionFileHistory
   set     currentRecordCount =
             (   select  count (*)
                 from    EDFacts_Submission.S052    ← Update
                 where   reportLevel = 'SCH' and schoolYear = @ReportingPeriod)
   where       specificationNumber = 'S052'    ← Update
           and reportingPeriod = @ReportingPeriod
           and reportLevel = 'SCH'
           and isMostCurrent = 'Y';

   /* LEA Level Record Count */
   update  EDFacts_Admin.SubmissionFileHistory
```

```
set     currentRecordCount =
          (   select  count (*)
              from    EDFacts_Submission.S052    ← Update
              where   reportLevel = 'LEA' and schoolYear = @ReportingPeriod)
where       specificationNumber = 'S052'    ← Update
        and reportingPeriod = @ReportingPeriod
        and reportLevel = 'LEA'
        and isMostCurrent = 'Y';

/* SEA Level Record Count */
update  EDFacts_Admin.SubmissionFileHistory
set     currentRecordCount =
          (   select  count (*)
              from    EDFacts_Submission.S052    ← Update
              where   reportLevel = 'SEA' and schoolYear = @ReportingPeriod)
where       specificationNumber = 'S052'    ← Update
        and reportingPeriod = @ReportingPeriod
        and reportLevel = 'SEA'
        and isMostCurrent = 'Y';
```

This routine needs to be adjusted to read from the correct submission table and to use the correct Submission file number. In not all three levels are being reported, then comment out or delete the appropriate section.

17. Create Submission Files Container

The container for the Create Submission files allows these processes to run in parallel.

Depending on the file type one or more of the individual file levels may not be included. Placing these in a container means they can be disabled or enabled as a group.

No edits are required from the template unless specific levels – SEA, LEA, or School – should be removed.

18. Create File Containers – One for Each Level

These are organizing containers that allow the individual levels to be written independently.

No edits are required from the template.

19. Get File Name for Level

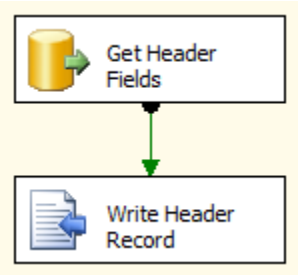
These Execute SQL Task processes get the file name and file path from the SubmissionFileHistory table. The values were set as part of the Set File History step #15.

```
select  top 1
        filename, filepath
from    EDFacts_Admin.SubmissionFileHistory
where       reportingPeriod = ?
        and specificationNumber = ?
        and reportLevel = 'SCH'   <-- set for the specific level in question
        and isMostCurrent = 'Y'
```

No edits are required from the template.

20. Write File Header Record

This is a data flow step with two tasks:



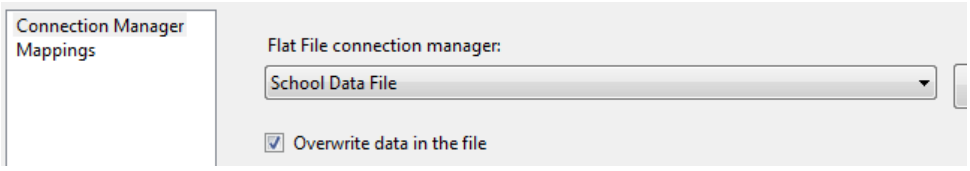
The Get Header Fields OLEBD Data Source task has a query that reads the appropriate information out of the SubmissionFileCharactersitics table and the SubmissionFileHistory table.

```
select  c.headerRecordFileType,
        h.currentRecordCount,
        h.fileName,
        h.fileIdentifier,
        h.reportingPeriod,
        ' ' as filler
from      EDFacts_Admin.SubmissionFileCharacteristic c
        join
          EDFacts_Admin.SubmissionFileHistory h
        on      c.reportingPeriod = h.reportingPeriod
           and c.specificationNumber = h.specificationNumber
           and c.reportLevel = h.reportLevel
where       c.reportingPeriod = ?
        and c.specificationNumber = ?
        and c.reportLevel = 'SCH'   ← set for the appropriate level
        and h.isMostCurrent = 'Y'
```

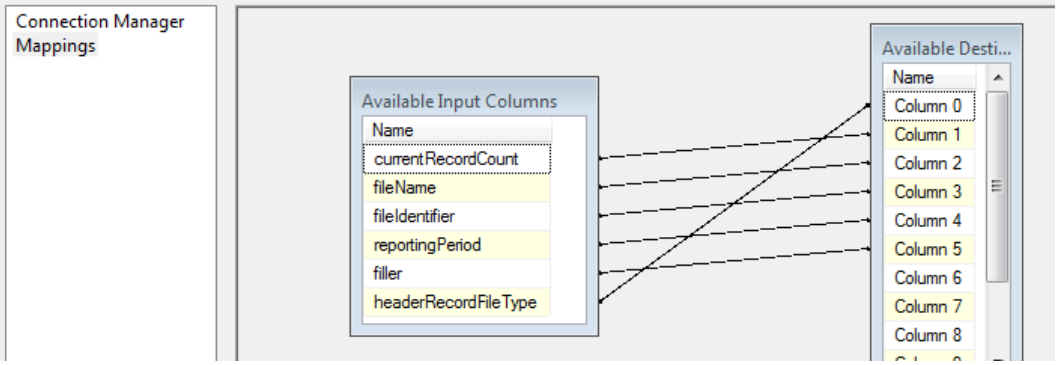
The parameters for this task are the ReportingPeriod and SpecificationNumber. The reportLevel filter needs to be set for the appropriate level in the three containers.

The Write Header Record simply writes the selected data to the first six columns in the appropriate data file connection.

On the Connection Manager screen, select the appropriate connection, State, LEA or School, and check the Overwrite data in the file checkbox.



Map the columns as required by the file specification.

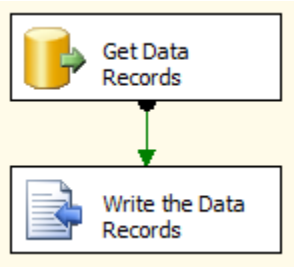


The header layout is the same for each of the file specifications.

No edits are required from the template.

21. Write File Data Records

This data flow task actually writes the data from the EDFacts_Submission table to the data file as required by the specification. The data flow consists of two tasks.



The first step reads the data records from the table and the second writes them to the data file.

The query in the Get Data Records OLEDB Data Source task is unique to each file specification. For the Membership file (S052) the query is:

```
select  ROW_NUMBER ()
          over (
            order by
              FIPS,
              stateAgencyNumber,
              stateLEAId,
              stateSchoolId,
              tableName,
              gradeLevel,
              raceEthnicity,
              gender,
              totalIndicator
          )
          as FileRecordNumber,
        FIPS,
        stateAgencyNumber,
        stateLEAId,
        stateSchoolId,
        tableName,
        gradeLevel,
        raceEthnicity,
```

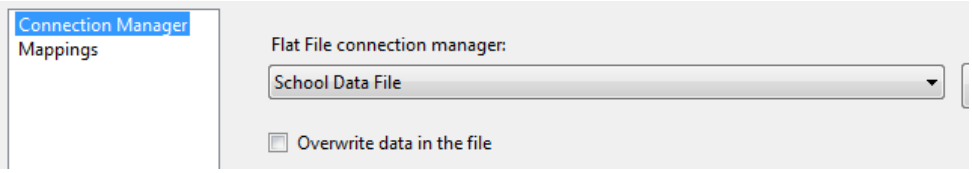
```
        gender,
        totalIndicator,
        explanation,
        totalCount
from    EDFacts_Submission.S052
where   reportLevel = 'SCH'
  and   schoolYear = ?
```

The first field in the query gets the unique line number for each record in the final file.

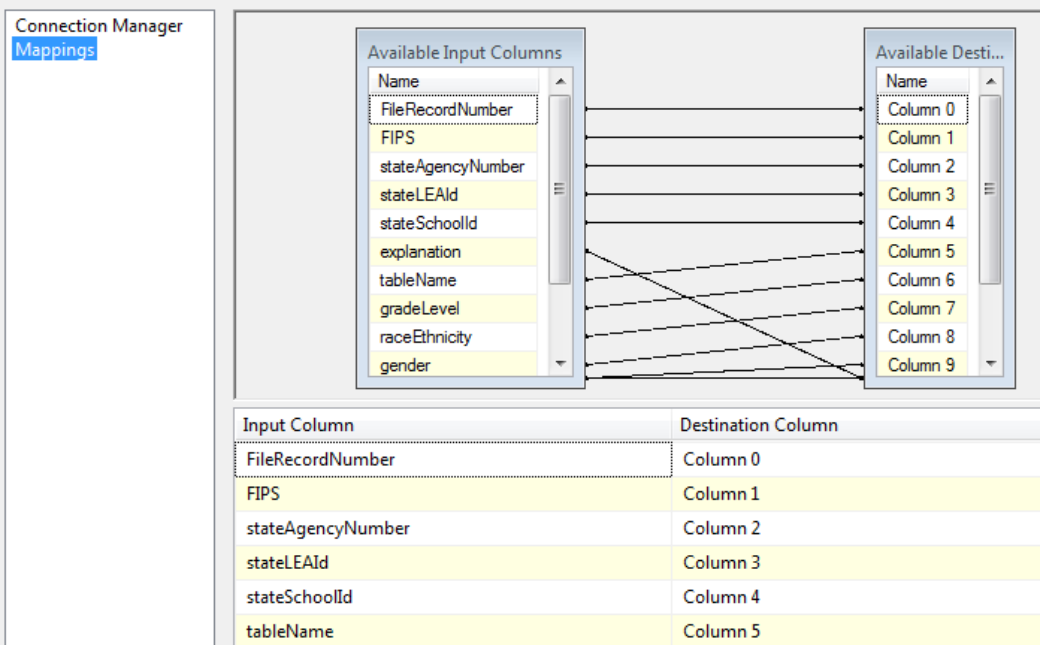
There is one parameter in this case, the school year or reporting period covered.

The reportLevel value in the WHERE clause needs to be set for each of the appropriate reporting levels.

The Write the Data Records Flat File Destination task then send the queried data to the appropriate file. Make sure the "Overwrite data in the file" checkbox is NOT checked – we want to append these data to the header record written in the previous step.



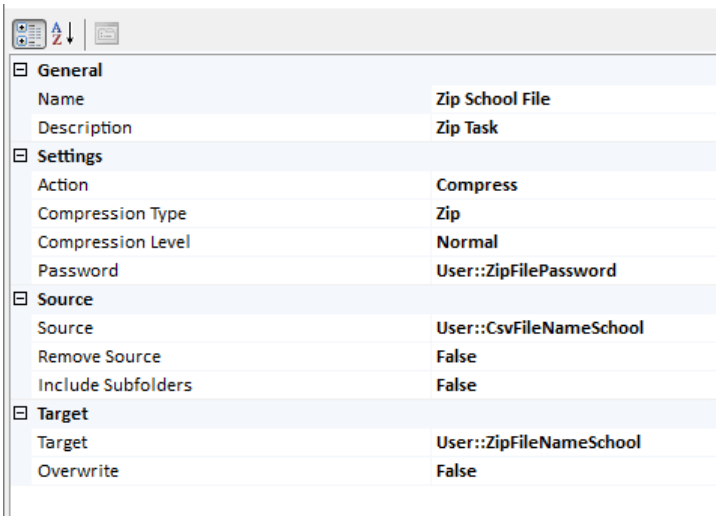
Map the fields to the appropriate columns in the submission file.



This task must be mapped specifically for each specification.

22. Zip Submission Files

The Zip files task zips the specific file for its level and stores it in the designated location. If a password has been specified for the file in the SubmssionFileCharacteristic table, then that password is applied to the zipped file.



No edits are required from the Template.

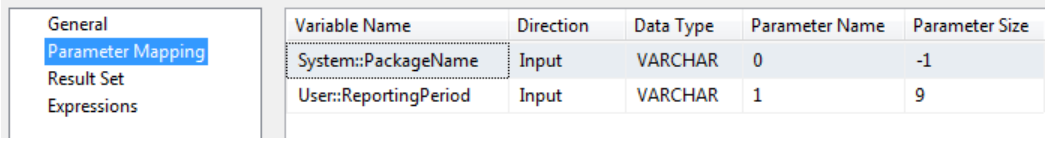
23. Set Success Email Fields

This task gets the To: and CC: lines that will be used in the success email.

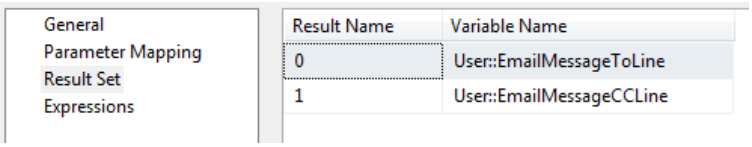
The query used is:

```
select  successNotificationToLine,
        successNotificationCCLine
from    EDFacts_Admin.ETLNotification
where       packageName = ?
        and reportingPeriod = ?
```

This routine takes two parameters, the Package name and the reporting period.



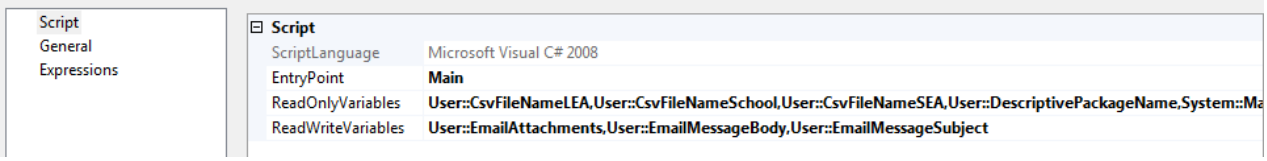
The results are written into the user variables for the email To: and CC: lines



No edits are required from the template.

24. Create Success Email Message Text

This task creates the email message text that will be sent and stores it in the EmailMessageBody user variable.



The script that does this task is:

```
public void Main()
{
  string strDescriptiveName;
  string strFileNameSchool;
  string strFileNameLEA;
  string strFileNameSEA;
  string strZipFileNameSchool;
  string strZipFileNameLEA;
  string strZipFileNameSEA;
  string strMsgText;
  string strAttachments;

  strDescriptiveName = (string)
      Dts.Variables["User::DescriptivePackageName"].Value;
  strFileNameSchool = (string)Dts.Variables["User::CsvFileNameSchool"].Value;
  strFileNameLEA = (string)Dts.Variables["User::CsvFileNameLEA"].Value;
  strFileNameSEA = (string)Dts.Variables["User::CsvFileNameSEA"].Value;
  strZipFileNameSchool = (string)Dts.Variables["User::ZipFileNameSchool"].Value;
  strZipFileNameLEA = (string)Dts.Variables["User::ZipFileNameLEA"].Value;
  strZipFileNameSEA = (string)Dts.Variables["User::ZipFileNameSEA"].Value;

  Dts.Variables["User::EmailMessageSubject"].Value =
      strDescriptiveName
      + " processng succeeded";

  strMsgText = (string)Dts.Variables["User::EmailMessageBody"].Value;
  strMsgText += "\n";
  strMsgText += "The " + strDescriptiveName;
  strMsgText += " data processing succeeded on ";
  strMsgText += Dts.Variables["System::MachineName"].Value;
  strMsgText += ".  \n\n";
  strMsgText += "The School file was written to: \n   ";
  strMsgText +=  strFileNameSchool;
  strMsgText += "\n\nThe LEA  file was written to: \n   ";
  strMsgText +=  strFileNameLEA;
  strMsgText += "\n\nThe SEA  file was written to: \n   ";
  strMsgText +=  strFileNameSEA;
  strMsgText += "\n\nZipped versions of these files have also been attached”;
  strMsgText += "to this email.";
  strMsgText += "\n\nTHEY HAVE NOT YET BEEN SUBMITTED TO USED";

  Dts.Variables["User::EmailMessageBody"].Value = (object)strMsgText;
  strAttachments = strZipFileNameSchool;
  strAttachments += "|";
  strAttachments += strZipFileNameLEA;
  strAttachments += "|";
  strAttachments += strZipFileNameSEA;

  Dts.Variables["User::EmailAttachments"].Value = (object)strAttachments;

  Dts.TaskResult = (int)ScriptResults.Success;
}
```

No edits are required from the template.

25. Global Container Completion

The Global Container exit constraint should be set as a "Completion" constraint as opposed to the default "Success" constraint. This ensures that we always fall through to the Update Email Log task.

To change the constraint, right click and select "Completion". The line should change to blue from green.

No edits are required from the template.

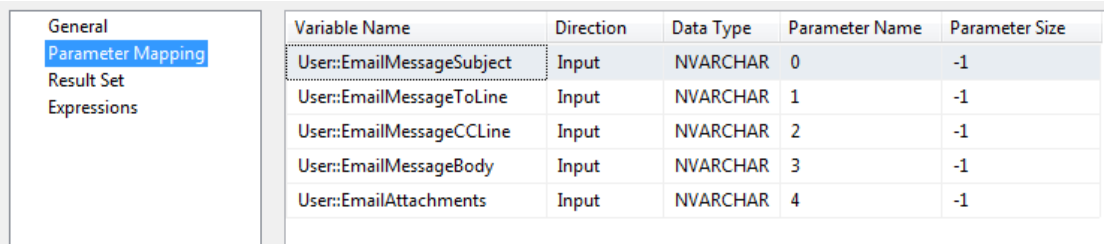
26. Update Email Log

Just prior to Sending the notification email and exiting, we write the email components to a log table. This way, if the email send process fails – bad address, size limit on the email server, etc. – we still have a record of the processing.

The Execute SQL task uses the following SQL Statement:

```
insert into  EDFacts_Admin.EmailLog (
               EmailDate,
               EmailSubject,
               EmailToLine,
               EmailCCLine,
               EmailMessageBody,
               EmailAttachmentList
             )
   values    (
               getdate (),
               ?,
               ?,
               ?,
               ?,
               ?
             )
```

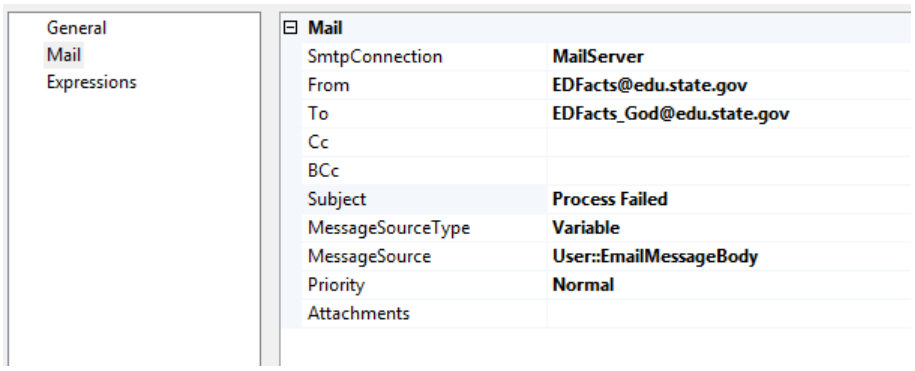
This routine takes the following five parameters and writes them to the log:



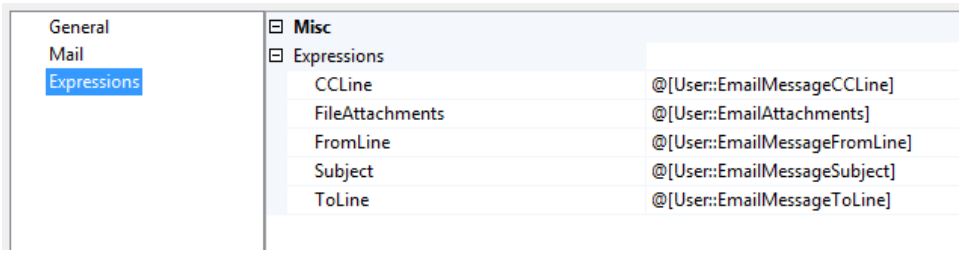
No edits are required from the template.

27. Send Notification

The final task is to email the notification to the appropriate folk. The basic set-up is as follows:



But the real work is in the Expressions that set the appropriate values for the email



No edits are required from the template.

Utility Routines

ef_Utility_BuildFileIdentifier

```
use [EDFacts];
GO

if exists
     (   select  *
         from    sys.objects
         where       object_id =
                       OBJECT_ID
(N'[EDFacts_Admin].[ef_Utility_BuildFileIdentifier]')
                 and type in (N'FN', N'FS', N'FT', N'TF', N'IF'))
  begin
drop function [EDFacts_Admin].[ef_Utility_BuildFileIdentifier];
  end
GO

set  ansi_nulls on;
GO
set  quoted_identifier on;
GO

create function [EDFacts_Admin].[ef_Utility_BuildFileIdentifier] ()
  returns varchar (32)
as
  /**
   * Function that builds a file identifier according to the EDFacts naming
   * conventions
   *
   *  @author      : Steven King, ESP Solutions Group
   *  @version     : 1.0  15-Dec-2011
   *  @returns     : varchar(32)    a file identifier consisting of date time and
   *               :                the user that created the file, e.g.:
   *               :                   28-Jan-2012 16:24 jYoung
   *  @system      : EDFacts Shared State Solution
```

```
 *  Notes        : builds the identifier using current datetime and system user.
 *               :
 * Revision History
 * -----------  ---------------  ------------------------------------------------
 * 15-Dec-2011   Steve King       Original draft function built
 */
begin
  declare @now   datetime
  declare @fileIdentifier   varchar (32)

  set @now = getdate ()

  set @fileIdentifier =
          right ('00'
                 + ltrim (rtrim (cast (datepart ("dd", @now) as char (2)))),
                 2)
        + '-'
        + case datepart ("mm", @now)
            when 1 then 'Jan'
            when 2 then 'Feb'
            when 3 then 'Mar'
            when 4 then 'Apr'
            when 5 then 'May'
            when 6 then 'Jun'
            when 7 then 'Jul'
            when 8 then 'Aug'
            when 9 then 'Sep'
            when 10 then 'Oct'
            when 11 then 'Nov'
            when 12 then 'Dec'
          end
        + '-'
        + right ('00'
                 + ltrim (rtrim (cast (datepart ("yyyy", @now) as char (4)))),
                 2)
        + ' '
        + right ('00'
                 + ltrim (rtrim (cast (datepart ("hh", @now) as char (2)))),
                 2)
        + ':'
        + right ('00'
                 + ltrim (rtrim (cast (datepart ("mi", @now) as char (2)))),
                 2)
        + ' '
        + +left (
            case charindex ('\', system_user)
              when 0
              then
                system_user
              else
                substring (system_user,
                           charindex ('\', system_user) + 1,
                           len (system_user)
                          )
            end,
            15)

  return @FileIdentifier
end
```

Note: individual states may have an alternative file identifier routine, in which case this routine will be modified.

ef_Utility_BuildFileName

```
use [EDFacts];
GO

if exists
     (   select  *
```

```
        from    sys.objects
        where       object_id =
                      OBJECT_ID (N'[EDFacts_Admin].[ef_Utility_BuildFileName]')
                and type in (N'FN', N'FS', N'FT', N'TF', N'IF'))
  begin
drop function [EDFacts_Admin].[ef_Utility_BuildFileName];
  end
GO

set  ansi_nulls on;
GO
set  quoted_identifier on;
GO

create function [EDFacts_Admin].[ef_Utility_BuildFileName] (
  @specificationNumber    varchar (8),
  @reportingPeriod        varchar (9),
  @reportingLevel         char (3),
  @FileVersion            char (7)
)
  returns varchar (25)
as
  /**
   * Function that builds a file name according to the EDFacts naming conventions
   *
   *  @author      : Steven King, ESP Solutions Group
   *  @version     : 1.0  15-Dec-2011
   *  @param       : @specificationNumber  The number of the EDFacts file in Sxxx
   *               :                       format
   *  @param       : @reportingPeriod      The school year for which data should
   *               :                       be processed in YYYY-XXXX format, for
   *               :                       example: 2009-2010
   *  @param       : @reportingLevel       The level for the file name: either
   *               :                       'SEA', 'LEA', or 'SCH'
   *  @param       : @fileVersion          the version for this instance of the
   *               :                       file.  This comes from the
   *               :                       IncrementFileVersion routine
   *  @returns     : varchar(25)           a filename in the EDFacts format:
   *               :                         <ss><LEV><tablename><Version>.tab
   *  @system      : EDFacts Shared State Solution
   *  Notes        : 'SELECTS' the appropriate components for the file name from
   *               :  the configuration tables.
   *               :
   * Revision History
   * ----------  ---------------  ------------------------------------------------
   * 15-Dec-2011   Steve King       Original draft function built
   */
  begin
    declare @Now   datetime
    declare @FileName   varchar (25)
    declare @PostalCode   char (2)
    declare @HeaderRecordFileName   varchar (9)

    set @Now = getdate ()
    set @PostalCode =
          (   select  top 1
                      postalCode
              from    EDFacts_Admin.StateConfig
              where   reportingPeriod = @reportingPeriod)
    set @HeaderRecordFileName =
          (   select  headerRecordFileName
              from    EDFacts_Admin.SubmissionFileCharacteristic
              where       reportingPeriod = @reportingPeriod
                      and specificationNumber = @specificationNumber
                      and reportLevel = @reportingLevel)
    set @FileName =
          @PostalCode
          + @reportingLevel
          + @HeaderRecordFileName
          + @FileVersion
          + '.tab'
```

```
    return @FileName
  end
GO
```

ef_Utility_IncrementFileVersion

```
use [EDFacts];
GO

if exists
     (   select  *
         from    sys.objects
         where       object_id =
                       OBJECT_ID
(N'[EDFacts_Admin].[ef_Utility_IncrementFileVersion]')
                 and type in (N'FN', N'FS', N'FT', N'TF', N'IF'))
  begin
drop function [EDFacts_Admin].[ef_Utility_IncrementFileVersion];
  end
GO

set  ansi_nulls on;
GO
set  quoted_identifier on;
GO

create function [EDFacts_Submission].[ef_Utility_IncrementFileVersion] (
  @FileReportingPeriod    as varchar (9),
  @SpecificationNumber    as varchar (8),
  @ReportingLevel         as char (3)
)
  returns varchar (7)
as
  /**
   * Function that builds the file version portion of an EDFacts submission file
   * name
   *
   *  @author      : Steven King, ESP Solutions Group
   *  @version     : 1.0  26-Jan-2012
   *  @param       : @FileReportingPeriod  The school year for which data should
   *               :                       be processed in YYYY-XXXX format, for
   *               :                       example: 2009-2010
   *               : @SpecificationNumber  The number of the specification in
   *               :                       question, in Sxxx format
   *               : @ReportingLevel       The three character indication of the
   *               :                       level for the
   *               :                       file: SEA, LEA, or SCH
   *  @returns     : 7 character identifier made up of MM, DD, HH, and a sequence
   *               : digit
   *  @system      : EDFacts Shared State Solution
   *  Notes        : Builds a file version for a submission file and report level.
   *               : The file version consists of the month, day, and hour that a
   *               : file is created plus a sequence digit.  The digit starts at 1
   *               : and increments as the report is regenerated with an hour.
   *               : The digit flips to "A" and increments through the alphabet if
   *               : more than 10 files are created within an hour.  We assume
   *               : there won't be more than 36 generations created within an
   *               : hour.
   *               : Process followed:
   *               :   1)  get the current date time
   *               :   2)  build the string MMDDHH from that
   *               :   3)  lookup in the table SubmissionFileHistory for this file
   *               :       with that portion of an identifier
   *               :   4)  either increment the sequence digit if it exists, or
   *               :       set the sequence digit value to 1 if new day and hour
   *               :       for that particular file and level
   *               :   5)  return the identifier value.
   * Revision History
   * -----------  ---------------  ------------------------------------------------
   * 28-Jan-2012  Steven King      Original draft version created
```

```
 */

begin
  declare @Identifier   varchar (7);
  declare @Now   datetime = getdate ();
  declare @SequenceDigit   char (1);

  set @Identifier =
          right ('00' + rtrim (cast (month (@Now) as char (2))), 2)
        + right ('00' + rtrim (cast (day (@Now) as char (2))), 2)
        + right ('00' + rtrim (cast (datepart (hour, @Now) as char (2))), 2)

     select          @SequenceDigit = right (fileVersion, 1)
     from            EDFacts_Admin.SubmissionFileHistory
     where               reportingPeriod = @FileReportingPeriod
                     and specificationNumber = @SpecificationNumber
                     and reportLevel = @ReportingLevel
                     and left (fileVersion, 6) = @Identifier;

  set @SequenceDigit =
        case
          when @SequenceDigit is null then '1'
          when @SequenceDigit = '9' then 'A'
          else char (ascii (@SequenceDigit) + 1)
        end;

  return @Identifier + @Sequencedigit;
end
```

Note: individual states may desire an alternate file versioning process, in which case this routine is customized.

SSIS Configuration

SSIS configuration information will be kept in the database table. In each environment we will need an environment variable to point to this table. See the section Individual Client Configuration below for table structure and environment variable set-up and definition.

SSIS Package Deployment

The SSIS packages will be deployed the SSIS database instance and executed from there.

Staging Table Design and Creation

Principles to table design

- Aligned with CEDS
- Lowest common record layout
- Submission file grouping

Staging Table Loading

SSIS Logging and Event Handling

There are two kinds of errors we need to manage and log.

The first are errors in the data that mean we cannot create the submission files or load the submission table. Those types of errors are managed in the data flow processes above. These are errors that the program managers and content experts correct.

The second kind are when the SSIS package fails for some reason, whether we did not account for some bad data and handle it appropriately or some other operational condition we have not accounted for. These are errors the SSIS package designer or programmers must correct.

When these latter errors occur, we want to know four things:

1. Which task failed
2. What error code is associated with the failure
3. What error message is associated with the failure
4. If associated with data, which row of data failed

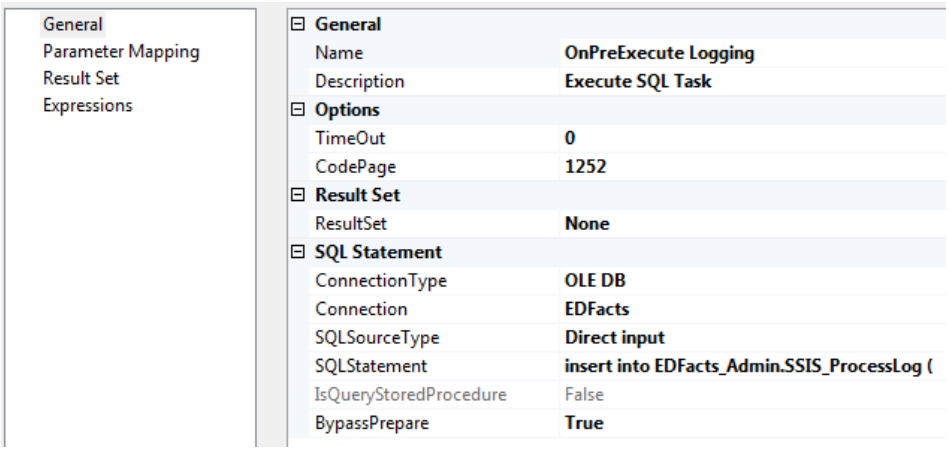
For these, we add two event handlers to the Generic ETL Package at the package level.

SSIS_ProcessLog Table

Field Name	Data Type	Description
eventID	int	An identity value for every event that is logged.
auditID	int	
executionInstanceGUID	uniqueIdentifier	A GUID for the specific running of the package
eventType	varchar(20)	The name of the event handler that wrote this record, either: ONError, OnPreExecute, or OnPostExecute
packageName	varchar(50)	The name of the package
taskName	varchar(50)	The name of the Task
taskID	uniqueIdentifier	The SSIS unique identifier for the Task
parentID	uniqueIdentifier	The SSIS unique Identifier for the Parent container – null for the package task
eventCode	int	Any error code for the task – 0 if no error
eventDescription	varchar(1000)	any Error message for the task, null if no error
taskStartTime	datetime	The date and time the event handler fired
taskStatus	varchar(50)	the overall status as logged by the event handler
hostMachineName	varchar(50)	the machine upon which the package is executing – note: this is not the DB against which the package is running necesarrily.

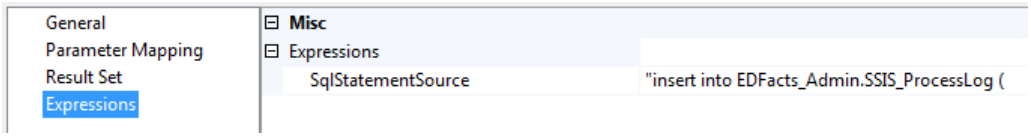
OnPreExecute Event Handler

The OnPreExecute event handler is a SQL task that populates a row in the SSIS_ProcessLog table for the start of every task in the package.



There are no parameter values passed into this routine nor any Result Sets coming out.

The SQL statement is built in an Expression.

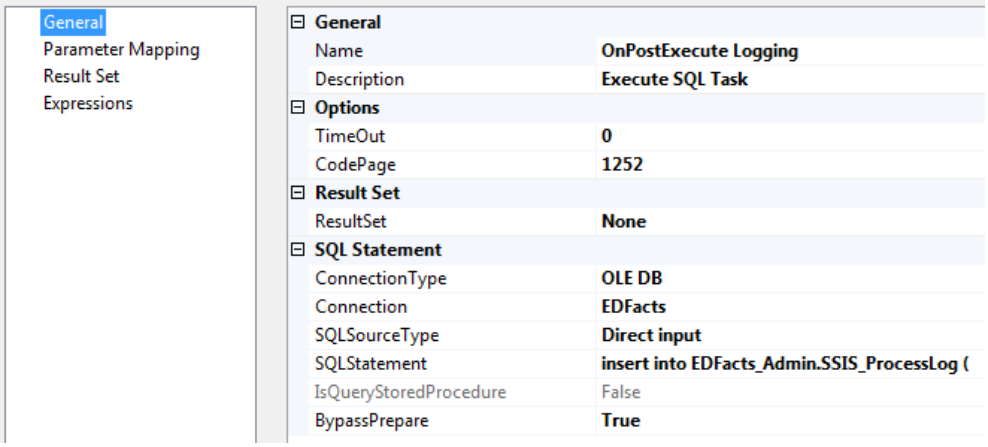


The expression is:

```
"insert into  EDFacts_Admin.SSIS_ProcessLog (
               executionInstanceGUID,
               eventType,
               packageName,
               taskName,
               taskID,
               parentID,
               eventCode,
               eventDescription,
               taskStartTime,
               taskStatus,
               hostMachine
             )
   values    (
               '" + @[System::ExecutionInstanceGUID] + "', --ExecutionInstanceGUID
               'OnPreExecute', --Event_Type
               '" + @[System::PackageName] + "', --Package_Name
               '" + @[System::SourceName] + "', --Task_Name
               '" + @[System::SourceID] + "', --Task_ID
               case
                 when '" + @[System::SourceParentGUID] + "' = '' then null
                 else '" + @[System::SourceParentGUID] + "'
               end, --Parent_ID
               0, --Event_Code
               '', --Event_Description
               GetDate (), --Task_Start_Time
               'Starting...', --Task_Status
               '" + @[System::MachineName] + "' --Host_Machine
             )"
```

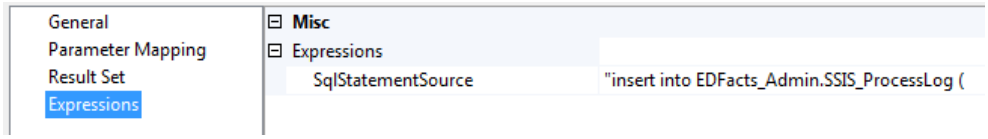
OnPostExecute Event Handler

The OnPostExecute event handler is a SQL task that populates another row in the SSIS_ProcessLog table at the end of every task in the package.



There are no parameter values passed into this routine nor any Result Sets coming out.

The SQL statement is built in an Expression.



The expression is:

```
"insert into  EDFacts_Admin.SSIS_ProcessLog (
               executionInstanceGUID,
               eventType,
               packageName,
               taskName,
               taskID,
               parentID,
               eventCode,
               eventDescription,
               taskStartTime,
               taskStatus,
               hostMachine
             )
   values    (
               '" + @[System::ExecutionInstanceGUID] + "', --ExecutionInstanceGUID
               'OnPostExecute', --Event_Type
               '" + @[System::PackageName] + "', --Package_Name
               '" + @[System::SourceName] + "', --Task_Name
               '" + @[System::SourceID] + "', --Task_ID
               case
                 when '" + @[System::SourceParentGUID] + "' = '' then null
                 else '" + @[System::SourceParentGUID] + "'
               end, --Parent_ID
               0, --Event_Code
               '', --Event_Description
               GetDate (), --Task_Start_Time
               'Complete', --Task_Status
               '" + @[System::MachineName] + "' --Host_Machine
             )"
```

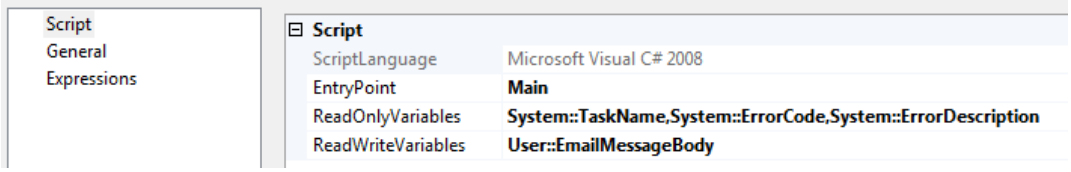
OnError *Event Handler*

The OnError event handler fires once for the current task that failed, and once more for each parent object, and finally also at the package level. Usually, only the first message gives enough information to be useful, but just to be sure we record them all.

The OnError event handler consists of two tasks:

One task updates the Email message body that an error occurred and the error code and description of the error. The second task writes a record into the SSIS_ProcessLog table with the error information.

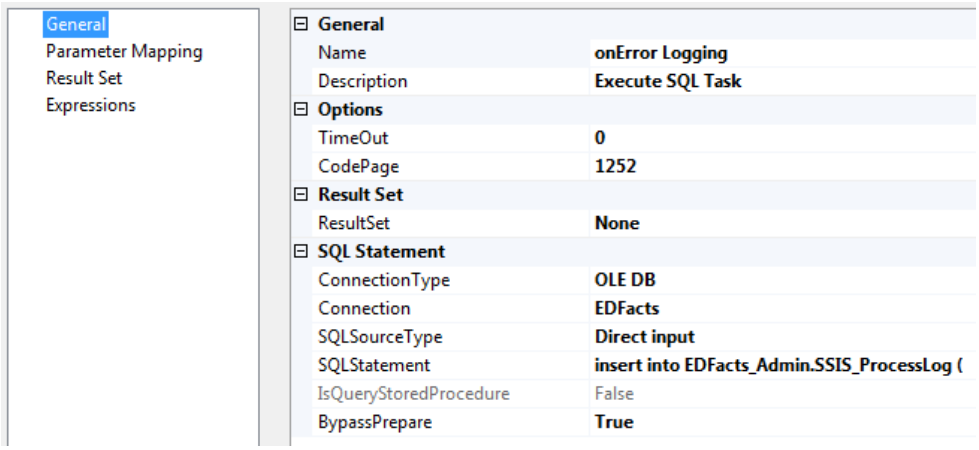
The Write Email message Body task is a Script task with three parameters coming in: the Task name, the error code, and the error description



The script appends this information to the EmailMessageBody user variable.

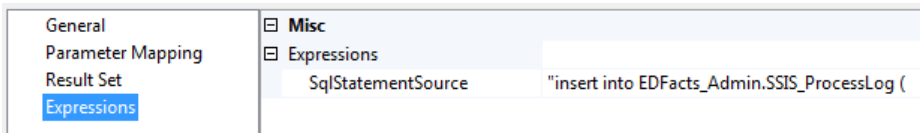
```
// onError Set Error Messages and Flags
public void Main()
{

    //Append error message to email error message variable for
    //email at end of package
    Dts.Variables["User::EmailMessageBody"].Value =
          Dts.Variables["User::EmailMessageBody"].Value
          + "\n\nTask Name:"
          + Dts.Variables["System::TaskName"].Value
          + "\n      Error Code: "
          + Dts.Variables["System::ErrorCode"].Value.ToString()
          + "\n      Error Description: "
          + Dts.Variables["System::ErrorDescription"].Value;
    Dts.TaskResult = (int)ScriptResults.Success;
}
```

The Error Logging task is a SQL task that inserts a record into the SSIS_ProcessLog table.



There are no parameter values passed into this routine nor any Result Sets coming out.

The SQL statement is built in an Expression.



The expression is:

```
"insert into  EDFacts_Admin.SSIS_ProcessLog (
               executionInstanceGUID,
               eventType,
               packageName,
               taskName,
               taskID,
               parentID,
               eventCode,
               eventDescription,
               taskStartTime,
               taskStatus,
               hostMachine
             )
   values    (
               '" + @[System::ExecutionInstanceGUID] + "', --ExecutionInstanceGUID
               'OnError', --Event_Type
               '"+ @[System::PackageName] + "', --Package_Name
               '"+ @[System::SourceName] + "', --Task_Name
               '"+ @[System::SourceID] + "', --Task_ID
               case
                 when '"+ @[System::SourceParentGUID] + "' = '' then null
                 else '"+ @[System::SourceParentGUID] + "'
               end, --Parent_ID
               '"+ (DT_STR, 15, 1252)@[System::ErrorCode] + "', --Event_Code
               '"+ @[System::ErrorDescription] + "', --Event_Description
               GetDate (), --Task_Start_Time
               'ErrorsOccurred', --Task_Status
               '"+ @[System::MachineName] + "' --Host_Machine
             )"
```

Validation Reports – Submission Tables

The US Education Department has documented a set of business rules that they use the validate EDFacts submissions. The business rules document is a spreadsheet. ED defines the columns in this spreadsheet as:

Below are the definitions of each column included in the spreadsheet portion of the guide.

- **Rule ID.** All business rules are assigned a unique ID number. You can use the Rule ID column to locate more information about edits triggered by a file submission. The letters at the start of the Rule ID indicate the type of edit.

Edits that start with "ER" are either format or validation edits.

Edits that start with an "M" are the edits that replaced CCD's match checks.[1]

Edits that start with an "S" are submission edits.

- **Error Type.** This column denotes the type of error that was found.

Format and Validation Errors. Format and validation errors both occur before the data are loaded into the staging database and are only reported through the Transmission Status Reports. Format errors occur when ESS cannot translate the file from its submitted format or cannot tell what format the file is in (xml, csv, txt, or tab). Validation errors usually identify invalid values when a permitted code set is provided.

Submission Errors. These errors occur in the staging database after the file has passed all format and validation edits. These errors ensure that submitted data meet or exceed an acceptable level of reasonability by checking the values entered in a field against other similar values in the same file or across files. They appear on the Submission Error Report and, for files that provide CCD data, on the Edit Reports.

Match Errors. These are a type of submission error and appear in the ESS match report (Submission Error Report page – Reports tab – Match Error Report row). They align with those formerly conducted by NCES in support of the Common Core Data (CCD) collection. All of these errors apply to file 029, Directory.

- **General Edit.** The general edit column denotes if the edit applies generally to more than one file specification. Because general edits apply to multiple files, these edits do not include a list of the associated file specifications. Examples of general edits are:

"ER-2 Format Error (Data is not in correct delimited (csv/tab) file format)" which applies to any csv or tab delimited file that comes in through ESS, and

"ER-37 Validation Error (The Category Code <value>, which was submitted for the reported <Table Type Name>, is not a Permitted Code)" applies to any file with specific permitted values.

Edit Type. This column tells you if the result of the edit is an error or a warning. Errors must be corrected. Once the error is corrected, it will no longer appear on the error report. Warnings should be investigated. If the data are determined to be incorrect, they should be corrected with a resubmission. If the data are determined to be correct, no update is needed.

Year to Year Change Edit. The year to year change flag lets you know if the edit compares prior year data to current year data.

Level (SEA, LEA, School). The reporting level to which the business rule applies - state education agency (SEA column), local education agency (LEA column), or school (School column). Some business rules apply to multiple levels. If, for instance, a business rule applies to the SEA file and the LEA file only, both the SEA and LEA columns will contain the value "Yes", but the School column will contain "No".

Error Message. This is the message text displayed on the ESS page or spreadsheet where the error or warning is provided.

Definition. The detailed description of the business rule including illustrative examples, where appropriate. Note that some rules have multiple components. That is, they apply to more than one data element on a single file. For instance, a file can be flagged with error ER-28 when either the mailing street address or the city is invalid in a submitted file. The Definition and Edit Logic can help you determine when this is true.

Edit Logic. The technical description of the business rule. This description includes the detailed logic employed in the business rule. Examples of values that display in this field include maximum values, checks for number of digits in a zip code, and comparisons of student counts.

Steward. The office responsible for the edit.

First ESS Release. Identifies the ESS version that first included the edit. This field helps users identify new edits and edits that will be implemented in a future release.

File Spec Used #1, #2…#7. Except for general edits, these columns identify the file(s) associated with the edit. Because some business rules draw on information from multiple EDEN files, several columns are needed to provide this information. For example, submission error S002-R17--'Children with disabilities student count represents more than 25% of total LEA student population'—uses data from several files. For this edit the File Spec Used columns lists files 002 (Children with Disabilities, School Age) and 089 (Children with Disabilities, Early Childhood) because the edit applies to the total number of children with disabilities reported on both of these files. It also lists file 052 (Membership) because the edit compares the total IDEA student count to the LEA's total student membership. Because edits sometimes use data from more than one file, sorting the spreadsheet by 'File Spec Used 1' will not always identify all the edits that apply to a specific file. Users should also perform the sort separately on each of the other file spec used columns.

December 2011 EDFacts 2011-11 Business Rules Guide Version 8.0 2

We are implementing these validation checks in the EDFacts Shared Solution.

Each individual rule or set of rules is implemented in its own stored procedure. The ES3 has an EDFacts_Submission.ValidationResults table that stores the results of the validation checks.

Pos	Column Name	Data Type
1	SubmissionNumber	varchar(8)

Pos	Column Name	Data Type
2	RuleNumber	varchar(9)
3	ReportingPeriod	varchar(9)
4	Severity	varchar(7)
5	LEA_ID	varchar(15)
6	SchoolID	varchar(15)
7	ErrorDescription	varchar(250)
8	AdditionalInfo	varchar(1000)

A sample routine is shown below. The routine selects any bad records and inserts them into the ValidationResults table. Each routine is named efbr_<submission number>_<rule number>. The "efbr" is short for "EDFacts Business Rule."

```
create procedure [EDFacts_Submission].[efbr_S052_R17]
  @ReportingPeriod as varchar (9)
as
  /**
   * Validates the S052 Membership file using the EDFacts System business rule
   * S052_R17.
   *
   *  @author      : Steven King, ESP Solutions Group
   *  @version     : 1.0  11-Jun-2012
   *  @param       : @ReportingPeriod  The school year for which data should be
   *               :                   processed in YYYY-XXXX format, for
   *               :                   example: 2009-2010
   *  @system      : State Collaborative EDFacts Management System
   *  returns      : Table with the following fields:
   *               :    Submission Number  - Always 'S052' in this file
   *               :    RuleNumber         - The EDFacts Business Rules Rule ID
   *               :    Severity           - The severity of the error - 'Error'
   *               :                         or 'Warning'
   *               :    LEA_ID             - State LEA ID, blank for SEA level
   *               :                         errors
   *               :    SchoolID           - State School ID, blank for SEA and
   *               :                         LEA level errors
   *               :    ErrorDescription   - A description of the error
   *               :    AdditionalInfo     - Additional information to spot the
   *               :                         specific problem
   *
   * Revision History
   * -----------  ---------------  ------------------------------------------------
   * 11-Jun-2012  Steven King      Original file created.
   */
  begin
    insert into  EDFacts_Submission.validationResults (
                  SubmissionNumber,
                  RuleNumber,
                  ReportingPeriod,
                  severity,
                  LEA_ID,
                  SchoolID,
                  ErrorDescription,
                  AdditionalInfo
                )
        /* S052-R17 - State Total Less than Sum of LEA totals*/

        select  'S052' as SubmissionNumber,
                'S052-R17' as RuleNumber,
                @ReportingPeriod as ReportingPeriod,
                'Error' as Severity,
                '' as LEA_ID,
                '' as SchoolID,
```

```
                'State total less than sum of LEA totals' as ErrorDescription,
                '' as AdditionalInfo
        from    EDFacts_Submission.S052
        where       reportLevel = 'SEA'
                and schoolYear = @ReportingPeriod
                and isnull (gradeLevel, '') = ''
                and isnull (raceEthnicity, '') = ''
                and isnull (gender, '') = ''
                and totalIndicator = 'Y'
                and totalCount <
                      (   select  sum (totalCount)
                          from    EDFacts_Submission.S052
                          where       reportLevel = 'LEA'
                                  and schoolYear = @ReportingPeriod
                                  and isnull (gradeLevel, '') = ''
                                  and isnull (raceEthnicity, '') = ''
                                  and isnull (gender, '') = ''
                                  and totalIndicator = 'Y')
  end

GO
```

Then for each submission file, there is a "master" store procedure that calls all the individual routines associated with that submission file. Each of these has a format similar to the following.

```
create procedure [EDFacts_Submission].[ef_Validation_S052]
  @ReportingPeriod as varchar (9)
as
  /**
   * Validates the S052 Membership file using the EDFacts System business rules.
   *
   *  @author      : Steven King, ESP Solutions Group
   *  @version     : 1.0  11-Jun-2012
   *  @param       : @ReportingPeriod  The school year for which data should be
   *               :                   processed in YYYY-XXXX format, for
   *               :                   example: 2009-2010
   *  @system      : State Collaborative EDFacts Management System
   *  returns      : Table with the following fields:
   *               :    Submission Number  - Always 'S052' in this file
   *               :    RuleNumber         - The EDFacts Business Rules Rule ID
   *               :    Severity           - The severity of the error - 'Error'
   *               :                         or 'Warning'
   *               :    LEA_ID             - State LEA ID, blank for SEA level
   *               :                         errors
   *               :    SchoolID           - State School ID, blank for SEA and
   *               :                         LEA level errors
   *               :    ErrorDescription   - A description of the error
   *               :    AdditionalInfo     - Additional information to spot the
   *               :                         specific problem
   *
   * Revision History
   * -----------  ---------------  ----------------------------------------
   * 11-Jun-2012  Steven King
   */
  begin
       delete from  edfacts_submission.validationResults
       where        submissionNumber = 'S052'
         and        reportingPeriod = @ReportingPeriod

    exec EDFacts_Submission.efbr_S052_R01_to_R16
      @ReportingPeriod = @ReportingPeriod
    exec EDFacts_Submission.efbr_S052_R17 @ReportingPeriod = @ReportingPeriod
    exec EDFacts_Submission.efbr_S052_R18 @ReportingPeriod = @ReportingPeriod
    exec EDFacts_Submission.efbr_S052_R19 @ReportingPeriod = @ReportingPeriod
    exec EDFacts_Submission.efbr_S052_R20 @ReportingPeriod = @ReportingPeriod
    exec EDFacts_Submission.efbr_S052_R21 @ReportingPeriod = @ReportingPeriod
    exec EDFacts_Submission.efbr_S052_R22 @ReportingPeriod = @ReportingPeriod
    exec EDFacts_Submission.efbr_S052_R23 @ReportingPeriod = @ReportingPeriod
```

```
  exec EDFacts_Submission.efbr_S052_R25 @ReportingPeriod = @ReportingPeriod
  exec EDFacts_Submission.efbr_S052_R28_to_R40
    @ReportingPeriod = @ReportingPeriod
  exec EDFacts_Submission.efbr_S052_R41 @ReportingPeriod = @ReportingPeriod
  exec EDFacts_Submission.efbr_S052_R42 @ReportingPeriod = @ReportingPeriod
  exec EDFacts_Submission.efbr_S052_R56 @ReportingPeriod = @ReportingPeriod
  exec EDFacts_Submission.efbr_S052_R60 @ReportingPeriod = @ReportingPeriod
end
```

Validation Reports – Staging Tables

System Monitoring and Management

Web Front-end

Individual Client Configuration

There are five tables in the EDFacts_Admin schema that maintain configuration information for individual clients.

After installing the database and copying the appropriate files into the right places, the data in these file tables will need to be edited and updated.

The beginning of this section describes each of those configuration tables. This followed by a section on the configuration of SSIS for a particular installation. A single state may have multiple installations, development and production for example.

The next section contains a checklist for the components that must be developed and customized for a particular client installation.

EDFacts_Admin.StateConfig Table

This table has a record for each school year (reporting period) with the details for this specific state.

Field Name	Data Type	Description
reportingPeriod	varchar(9)	The 9 character representation of the school year or reporting period represented. Formatted as YYYY-XXXX, for example '2011-2012'
stateName	varchar(20)	The name of the state
postalCode	char(2)	The state's 2 character USPS abbreviation
stateFIPSCode	char(2)	The two-digit Federal Information Processing Standards (FIPS) for the state
stateAgencyNumber	char(2)	The State Agency Identifier is assigned by ESS and the only valid value currently available is "01" for the SEA. In the future, additional state agencies may be able to submit data directly to ESS and they will receive different State Agency Identifiers. This ID cannot be updated through this file.

Field Name	Data Type	Description
storageDirectoryRootPath	varchar(200)	The fully qualified path to the root working directory for files. Drive letters are mapped on the system where the SSIS packages are executed – i.e. the database server.
emailFromLine	varchar(100)	the address to be used for the "From:" line in notification emails
isCharterAllowed	char(1)	A Y/N flag indicating if charter schools are an allowable option in the directory file
isUngradedAllowed	char(1)	A Y/N flag indicating if "Ungraded" is an allowable option for this state
stateLogoMIMEType	varchar(25)	The image type of the state Logo, e.g. "image/jpeg" or "image/gif"
stateLogo	varbinary(8000)	The state Logo image binary data

EDFacts_Admin.StateCharacteristic Table

This table has a records for each school year (reporting period) with the characteristics for the specific state stored as name/value pairs. The table includes characteristics like whether the state hasCharterSchools or whether grades 13 or Ungraded should be included as options.

Field Name	Data Type	Description
reportingPeriod	varchar(9)	The 9 character representation of the school year or reporting period represented. Formatted as YYYY-XXXX, for example '2011-2012'
name	varchar(100)	
value	varchar(100)	

EDFacts_Admin.SubmissionFileCharacteristic Table

This table has configuration information for each submission file and reporting period. These data are used when creating the submission files.

Field Name	Data Type	Description
reportingPeriod	varchar(9)	The 9 character representation of the school year or reporting period represented. Formatted as YYYY-XXXX, for example '2011-2012'
specificationNumber	varchar(8)	The reference used for the specification number, also the name of the table in the EDFacts_Submission schema. Most specs its simple the spec number like "S052". In some cases, the different levels are enough different wo warrant their own table, like "S029_LEA"
reportLevel	char(3)	SEA, LEA, or SCH
headerRecordFileName	varchar(9)	the portion of the filename in the header rec specific to this file
headerRecordFileType	varchar(150)	The filetype field of the header rec for this specificatrion
dataRecordTableName	varchar(20)	The table name to be used in the data records for this file

Field Name	Data Type	Description
defaultFilePath	varchar(100)	The default directory location where the submission files are to be stored – relative to the storageDirectoryRootPath from the stateConfig table.
lastRunDate	datetime	The date and time this submission file was last created.

Prior to 2011-12, ED had separate specification document for CSV files and XML files and referred to the files as with Nxxx or Xxxx depending on whether the file was Non-XML or XML. Beginning with the 2011-12 school year, they combined the specification documents and named them Cxxx. To avoid confusion (or perhaps to contribute to it) this EDFacts system refers to the submission files as Sxxx.

EDFacts_Admin.ETLNotification Table

The notification sent at the end of each package execution can be configured to go to a different set of users. Error notification can be sent to a different set than the success email.

The header information for the notification emails for each of the ETL packages in this system is defined in the following table.

Field Name	Data Type	Description
reportingPeriod	varchar(9)	The 9 character representation of the school year or reporting period represented. Formatted as YYYY-XXXX, for example '2011-2012'
packageName	varchar(50)	
successEmailSubjectLine	varchar(100)	
errorEmailSubjectLine	varchar(100)	
errorNotificationToLine	varchar(250)	
errorNotificationCCLine	varchar(250)	
successNotificationToLine	varchar(250)	
successNotificationCCLine	varchar(250)	

All of these emails will come "From" the address configured in the stateConfig table. The email server connection information to be used is configured in the SSIS_Configuration table.

There should be one line in this table for each SSIS package and reportingPeriod

EDFacts_Admin.StateCodeTranslation Table

This table is used to map the state codes to the codes used in the EDFacts submission files. As the submission files are created, a code translation takes place using the information from this table.

Field Name	Data Type	Description
reportingPeriod	varchar(9)	The 9 character representation of the school year or reporting period represented. Formatted as YYYY-XXXX, for example '2011-2012'

Field Name	Data Type	Description
codeSetName	varchar(20)	The name of the code set being translated. This name should be unique in the system for a given reporting period
stateCode	varchar(25)	The state code found in the staging tables and or source system
EDFactsCode	varchar(15)	The EDFacts system code to be used in the submission file

EDFacts_Admin.SSIS_Configuration Table

This is the standard SSIS configuration table that SSIS uses under database configuration.

Field Name	Data Type	Description
ConfigurationFilter	nvarchar(255)	
ConfiguredValue	nvarchar(255)	
PackagePath	nvarchar(255)	
ConfiguredValueType	nvarchar(20)	

EDFacts System SSIS Environment Variable

Finally, there must be an environment variable on the machine where the SSIS packages execute pointing to the EDFacts_Admin.SSIS_Configuration Table. This variable is name EDFactsSSISConfiguration and contains as a value <what>

Individual Client Development and Customization Checklist

- File identifier routine
- File version routine
-

Managing Client Contributions

<Insert description of how we will manage client contributions to the EDFacts Shared State Solution >

Development Principles

Common package variables and initiation

Script to load the package variables with the default/initial values

Data validation decision/logging

Generic data validation log layout and process

Process result email notification

Get to and cc lines from table for this ETL package (default option in table if package name not found)

Build the message body

Convention for message subject line

Standards and Best Practices

Naming conventions

T-SQL procedure naming and comment conventions

Development Team Peer Code Reviews

Client Design Reviews