

Development Guide

ED*Facts* Shared State Solution



ESP Solutions Group

Authors: Steven King
Date: 19 July 2016
Version: 0.3

Change History Log

Date	Pages	Summary of Changes	Version	Authorized By
5-Jan-2012		Initial Draft	v0.1	Steven King
6-Aug-2014		Update staging package generic design, specifically the Post Process procedure design and name; and the email generation. Email now includes the staged table name and the reporting period processed.	v0.2	Steven King
19-July-2016		Adding a new 'Client Set Up' section to the document.	V0.3	Kathleen Browning

Table of Contents

Table of Figures.....	iv
Screen Shots.....	iv
Introduction.....	1
System Overview.....	2
Objectives	3
Scope	3
Definitions	3
Technologies.....	3
Relation to the CEDS.....	4
Design Principles	4
Development Plan	5
Client Setup Procedures	5
Staging Table Design and Loading	6
Principles to table design	6
Generic Stage Loading Package Connection Managers	6
Generic Stage Loading Package ETL Variables	7
Generic Stage Loading Package ETL Flow Design	9
Submission Table Load and File Creation	24
Submission Table Design.....	24
Generic Submission Package Connection Managers	24
Generic Submission Package ETL Variables	27
Generic Submission Package ETL Flow Design	29
Utility Routines	54
Generic Package Configurations	58
“Core Connections” Package Configuration.....	60
“Package Specific” Package Configuration	60
SSIS Package Deployment.....	60
SSIS Logging and Event Handling.....	60
SSIS_ProcessLog Table.....	60
OnPreExecute Event Handler	61
OnPostExecute Event Handler.....	62
OnError Event Handler.....	63
Validation Reports – Submission Tables	65
Validation Reports – Staging Tables	70
System Monitoring and Management	70
Web Management System	70
Individual Client Configuration.....	70
Client Configuration Data Tables.....	70
EDFacts_Admin.StateConfig Table	70
EDFacts_Admin.StateCharacteristic Table.....	71
EDFacts_Admin.SubmissionFileCharacteristic Table.....	71
EDFacts_Admin.StateCodeTranslation Table.....	72
EDFacts_Admin.SSIS_Configuration Table	72
Managing State Code Set Translation Values.....	72
Individual Client Development and Customization Checklist.....	74
Managing Client Contributions	74

Standards and Best Practices.....	74
Naming conventions	74
T-SQL procedure naming and comment conventions	74

Table of Figures

Figure 1: Stage Loading Generic Variable List.....	7
Figure 2: Generic Stage Loading Package Control Flow	9
Figure 3: Stage Loading, Database Source Data, Typical Data Flow	12
Figure 4: Stage Loading, Excel Source Data, Typical Data Flow.....	15
Figure 5: Submission Loading Generic Package Control Flow	29
Figure 6: Submission Loading, Validate Source Data, Data Flow	33

Screen Shots

Screen 1: Stage Loading, Generic Connection Managers	6
Screen 2: Stage Loading Package, Set Package Defaults, General Information	10
Screen 3: Stage Loading Package, Set Package Defaults, Parameter Mapping	11
Screen 4: Stage Loading Package, Set Package Defaults, Result Set	11
Screen 5: Stage Loading Package, Delete Previously Staged Data, General Information.....	11
Screen 6: Stage Loading Package, Delete Previously Staged Data, Parameter Mapping	12
Screen 7: Stage Loading, Source Data from Database, Read Source Records, Connection Manager.....	13
Screen 8: Stage Loading, Source Data from Database, Count Records	14
Screen 9: Stage Loading, Source Data from Database, Write Staging Records, Connection Manager.....	14
Screen 10: Stage Loading, Source Data from Database, Write Staging Records, Column Mapping.....	15
Screen 11: Stage Loading, Excel Data Source, Project Properties	16
Screen 12: Stage Loading, Excel Source Data, Excel Source Connection, Connection Manager.....	17
Screen 13: Stage Loading, Excel Source Data, Excel Source Connection, Columns	17
Screen 14: Stage Loading, Excel Source Data, Data Conversion	18
Screen 15: Stage Loading, Excel Data Source, Count Records.....	18
Screen 16: Stage Loading, Excel Data Source, Write Stage Records, Connection Manager	19
Screen 17: Stage Loading, Excel Source Data, Write Stage Data, Mappings	19
Screen 18: Stage Loading, Post Load Processing, General Information	20
Screen 19: Stage Loading, Post Load Processing, Parameter Mapping.....	21
Screen 20: Stage Loading, Update Email Log, Parameter Mapping	23
Screen 21: Stage Loading, Send Notification, Mail Settings.....	23
Screen 22: Stage Loading, Send Notification, Expressions.....	24
Screen 23: Submission Loading, Generic Connection Managers	25
Screen 24: Submission Loading, Connections, Mail Server	25

Screen 25: Submission Loading, Set Package Defaults, General Information	30
Screen 26: Submission Loading, Set Package Defaults, Parameter Mapping.....	31
Screen 27: Submission Loading, Set Package Defaults, Result Set.....	31
Screen 28: Submission Loading, Check-Create Directories, Script Settings	31
Screen 29: Submission Loading, Clear Previous Invalid Records, General Information.....	32
Screen 30: Submission Loading, Clear Previous Invalid Records, Parameter Mapping	32
Screen 31: Submission Loading, Validate Source Data, Columns.....	37
Screen 32: Submission Loading, Validate Source Data, Bad Records Count.....	37
Screen 33: Submission Loading, Validate Source Data, Write Invalid Records, Connection Manager	38
Screen 34: Submission Loading, Validate Source Data, Write Invalid Records, Field Mapping	38
Screen 35: Submission Loading, Get Invalid Records File Name, General Information	39
Screen 36: Submission Loading, Get Invalid Records File Name, Parameter Mapping	40
Screen 37: Submission Loading, Get Invalid Records File Name, Result Set	40
Screen 38: Submission Loading, Get Invalid Records Data Flow	40
Screen 39: Submission Loading, Get Invalid Records, Mappings.....	41
Screen 40: Submission Loading, Zip Invalid Records File	41
Screen 41: Submission Loading, Create Error Email Message, General Settings.....	42
Screen 42: Submission Loading, Conduct the ETL, General Information.....	45
Screen 43: Submission Loading, Conduct the ETL, Parameter Mapping	45
Screen 44: Submission Loading, Set File History, General Information	45
Screen 45: Submission Loading, Set File History, Parameter Mapping	46
Screen 46: Submission Loading, Write File Header, Data Flow	48
Screen 47: Submission Loading, Write File Header, Connection Manager	48
Screen 48: Submission Loading, Write File Header, Mappings	48
Screen 49: Submission Loading, Write File Data Records, Data Flow.....	49
Screen 50: Submission Loading, Write File Data Records, Connection Manager.....	50
Screen 51: Submission Loading, Write File Data Records, Mappings.....	50
Screen 52: Submission Loading, Zip Submission Files	51
Screen 53: Submission Loading, Create Success Email Message General Information	51
Screen 54: Submission Loading, Update Email Log, Parameter Mapping	54
Screen 55: Submission Loading, Send Notification, Mail Settings.....	54
Screen 56: Submission Loading, Send Notification, Expressions.....	54
Screen 57: Package Configuration Organizer.....	60
Screen 58: Logging and Event Handling, OnPreExecute Event, General Information.....	61
Screen 59: Logging and Event Handling, OnPreExecute Event, Expressions.....	61
Screen 60: Logging and Event Handling, OnPostExecute Event, General Information	62
Screen 61: Logging and Event Handling, OnPostExecute Event, Expressions	62
Screen 62: Logging and Event Handling, OnError Event, Data Flows	63
Screen 63: Logging and Event Handling, OnError Event, Update Email Script Information	63
Screen 64: Logging and Event Handling, OnError Event, General Information	64

Screen 65: Logging and Event Handling, OnError Event, Expressions	64
---	----

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. Many EDFacts coordinators put something together that they can use, with little thought about sharing with others – no need, nor with thought toward building robust monitoring and notification facilities. These state systems often have sketchy or missing documentation.

Many states are having to redesign or rebuild 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.

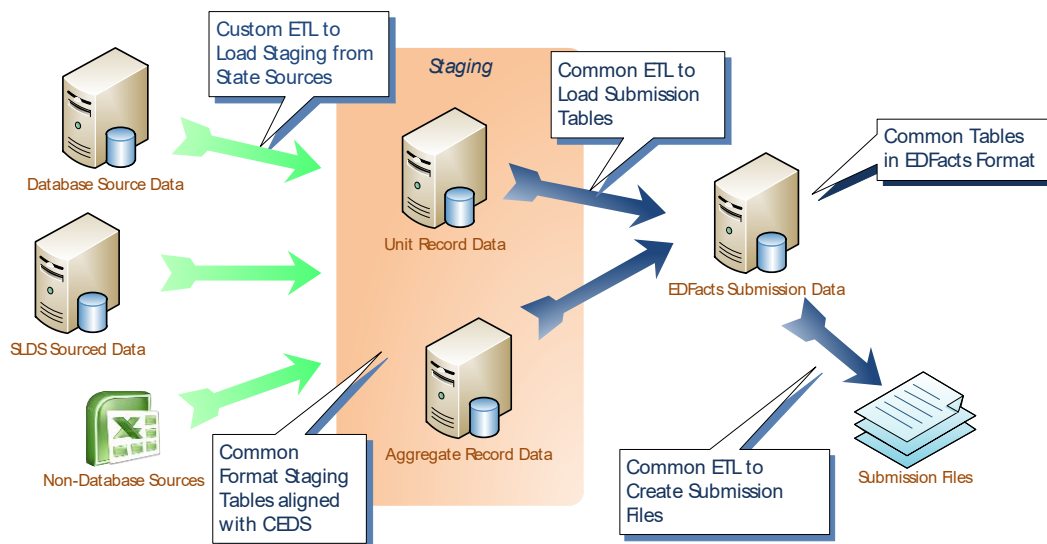
States can get and modify these processes for their custom needs, or engage ESP to do the customization for them.

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.

ES3 builds a set of tables that mirror the structure of those files. These 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 are used to construct the EDFacts Submission tables.

To the maximum degree possible, the ES3 standardizes the staging tables across states. Some states 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 aligns the staging tables with the Common Education Data Standards where that is possible. ES3 does use state codes for individual elements in the staging tables.

By standardizing the staging table design, the ETL process to load the EDFacts Submission tables is common across all state clients.

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 these database tables 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 is made to standardize where possible and to build templates for each data load.

The staging load ETL processes still include standard 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) **Developers** – to provide guidance for their work and the processes to use
- 2) **State EDFacts and Program Staff** – 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. Nor does this document go into the details of using Visual Studio, the design of SSIS packages, nor SQL development.

Definitions

<to be expanded>

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, ES3 only requires read-only access to source data.
- The system writes to the file system only to write submission files or invalid record reports
- The system only reads from the file system to retrieve data from non-database sources, e.g. Excel
- Core connection information is shared across all packages, i.e. the location of the EDFacts Database, and email server configuration
- Custom code is encapsulated in stored procedures to the maximum degree practicable
- Users are notified of execution results via email
- Standard Microsoft SQL Server tools are used – no proprietary tools, add-ons, or components
- The system will be designed and built with easy maintenance and modification as goals

Development Plan

ES3 is organized in two Visual Studio projects, one for staging table loads, and one for submission table loads and file creation. The Stage Loading project is custom for each state. The Submission Loading project components, on the other hand, are common across all states.

Client Setup Procedures

1. We will need a SQL database named “EDFacts”. This can be on an existing data base server. Let us know what version: 2008R2, 2012, 2014, etc.
2. We will need an instance of SQL Integration services (SSIS). This can be on the same server as the one above or separate (same is preferred, but it doesn’t really matter.)
3. We will create SSIS packages that do the necessary ETL. The packages will be stored in the SSIS repository on the SSIS server. Depending on the version of SQL, this will be either the MSDB or SSISDB database on the SSIS Server.
4. We will customize the web front end application to run on an IIS server of your choice and integrate with your security. Our web app will need access to the SQL servers above.
5. The SSIS SQL Server will need to have SQL Agent running. We will need to be able to create Agent jobs that have a single step of running one of the SSIS packages from the repository above. The SQL agent can use either our web app’s Service account or we can create a Proxy account. The credential associated with the proxy, or the web app service account will require all the necessary SQL access to query source systems and write to the EDFacts database.
6. We need a file server directory location where we will create and save the EDFacts submission files. This directory must be accessible by the SSIS packages we run.
7. We will need workstations where the ESP staff can do our development work. We will want two of these. These can be virtual machines. They will need to have the appropriate version of Visual Studio (2008, 2012, 2014, etc.) and SQL Management Studio. They need a copy of Word and Excel. There are some additional development tools that ESP will need to install on these- we have those tools licensed for our use. These workstations need access to the internet via port 80 to get to our TFS and for general internet queries.
8. We will use whatever VPN/RDC process you want us to use to remotely access these boxes. We want to keep all data and query results inside your network.
9. The SQL servers and the work stations must be able to query the source data systems that hold EDFacts data.
10. Start pulling together a list of the EDFacts submissions and where you currently get the data. We can send you an Excel template if you don’t have something already. If you have stored procedures, queries, or other processes documented then start pulling that information together for us as well.

If you want separate Dev/Test/Prod environments, then multiply the above as appropriate. Dev and Prod is probably sufficient. Building three environments for one or two WDE users seems like overkill.

In the meantime, on our side we will:

1. Get a SharePoint project site set up
2. Build a loose project plan into which we will need to add dates

3. Get our tracking sheets together
4. Get a clean project built in TFS for your state that has all the Stage-to-submission packages ready to deploy

Staging Table Design and Loading

<description>

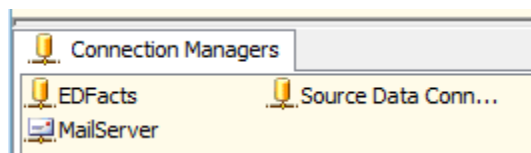
Principles to table design

When the staging tables are put together for a state, the following set of design principles are followed.

- Aligned with CEDS – the fields in the staging tables are aligned with CEDS to the maximum degree possible.
- Submission file grouping – submission files have been grouped and the staging tables designed to maximize their utility.
- State Specific fields for auditing/data review – states often have information in their source data that are used to populate the standard staging tables. We can add these custom fields to the end of the staging table and use them during the stage data post processing to set the standard values. These fields are not referenced during the Submission Loading routines.
- EDFacts_LocalSource Schema – ES3 has a schema that holds data from Excel, text files, and other non-database sources. These tables have the same structure and layout as the original file source. The staging process loads these first with little data manipulation, then loads the staging table from there.

Generic Stage Loading Package Connection Managers

There are typically three connections used by a Stage Loading ETL package.



Screen 1: Stage Loading, Generic Connection Managers

EDFacts connects to the EDFacts data base and is the destination for data in the Stage Loading packages.

Source Data Connection is the read only connection to the source system tables or files for this package. This can be an ODBC connection for data coming from a

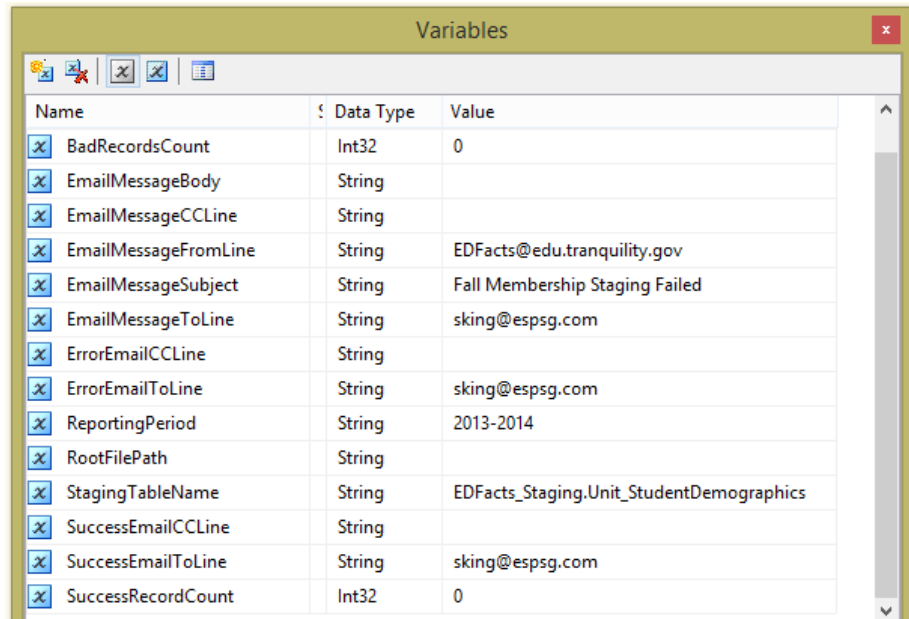
database, or it can be a file data source for csv or other delimited text data. This can be an Excel or Access data source if data are coming from those applications.

The Source Data Connection can be renamed to something more meaningful as needed.

The MailServer connection points to the email server from which the package will send the summary email.

Generic Stage Loading Package ETL Variables

At minimum, a Stage Loading package will have the following variables.



Name	Data Type	Value
BadRecordsCount	Int32	0
ErrorMessageBody	String	
ErrorMessageCCLine	String	
ErrorMessageFromLine	String	EDFacts@edu.tranquility.gov
ErrorMessageSubject	String	Fall Membership Staging Failed
ErrorMessageToLine	String	sking@espsg.com
ErrorEmailCCLine	String	
ErrorEmailToLine	String	sking@espsg.com
ReportingPeriod	String	2013-2014
RootFilePath	String	
StagingTableName	String	EDFacts_Staging.Unit_StudentDemographics
SuccessEmailCCLine	String	
SuccessEmailToLine	String	sking@espsg.com
SuccessRecordCount	Int32	0

Figure 1: Stage Loading Generic Variable List

These variables and their meaning are:

Variable	Value Source	Used By
BadRecordsCount	Originally zero and may be updated by the Data Flow or Stage Data clean-up task	
ErrorMessageBody	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
ErrorMessageCCLine	Set when the Email is created either as ErrorEmailCCLine or SuccessEmailCCLine	send notification
ErrorMessageFromLine	Set in Set Package Defaults. Read from StateConfig	send notification

Variable	Value Source	Used By
EmailMessageSubject	Default value of "<package name> Failed" is set in the package. Value gets updated in [Create Email Message Text] or [Create Success Email Text] tasks	send notification
EmailMessageToLine	Set when the Email is created either as ErrorMessageToLine or SuccessEmailToLine	send notification
ErrorEmailCCLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Email Message Text
ErrorEmailToLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Email Message Text
ReportingPeriod	the reporting period covered by the data for which this staging routine applies	all the steps
RootFilePath	read in from the state config table in the Set Filepath routine	send notification
StagingTableName	the name of the staging table into which these data are written	Create Email Message Text
SuccessEmailCCLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Email Message Text
SuccessEmailToLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Email Message Text
SuccessRecordsCount	The number of records read from the source data. Set in the Copy source records data flow task	Create Email Message Text

Generic Stage Loading Package ETL Flow Design

The generic stage loading package ETL flow process is shown below

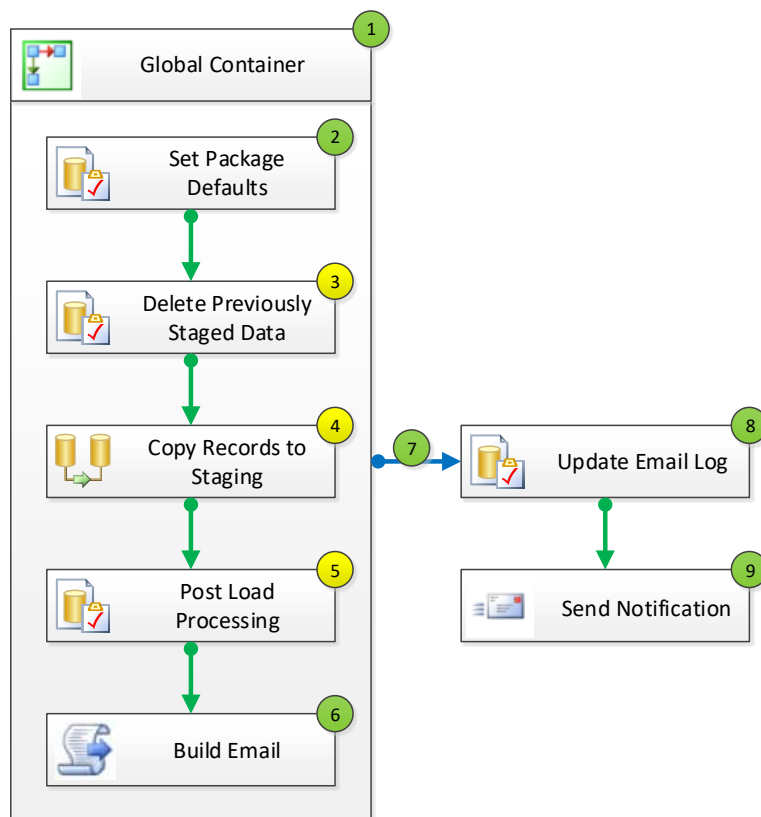


Figure 2: Generic Stage Loading Package Control Flow

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 stage loading package. Only the yellow numbered tasks need to be modified for a specific EDFacts staging table, specifically:

- Delete Previously Staged Data (#3)
- Copy Records to Staging (#4)
- Post Load Processing (#5)

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

The first real task of the standard Stage Loading package is one that sets some of the package variable defaults from the configuration data in the EDFacts_Admin tables.

General	
Name	Set Package Defaults
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
Result Set	
ResultSet	Single row
SQL Statement	
ConnectionType	OLE DB
Connection	EDFACTS
SQLSourceType	Direct input
SQLStatement	select cfg.storageDirectoryRootPal...
IsQueryStoredProcedure	False
BypassPrepare	True

SQLStatement
Specifies the query to be run by the task.

Browse... Build Query... Parse Query

Screen 2: Stage Loading Package, Set Package Defaults, General Information

This is an Execute SQL Task. The ResultSet is a single row of data. Set the connection to the EDFacts connection. The SQL Statement to store is:

```
select
    storageDirectoryRootPath,
    emailMessageFromLine
from EDFacts_Admin.StateConfig
where reportingPeriod = ?
```

The question mark at the end indicates a parameter that will be passed to the query at runtime.

Tell the system about package parameters using the Parameter Mapping menu option.

General	Variable Name	Direction	Data Type	Parameter Name	Parameter Size
Parameter Mapping	User::ReportingPeriod	Input	VARCHAR	0	9
Result Set					
Expressions					

Screen 3: Stage Loading Package, Set Package Defaults, Parameter Mapping

In this example, map parameter 0 (the first “?” in the query – parameters use zero-based numbering) to the variable User::ReportingPeriod.

The next step is to do something with the single row of data we receive.

General	Result Name	Variable Name
Parameter Mapping	0	User::RootFilePath
Result Set	1	User::EmailMessageFromLine
Expressions		

Screen 4: Stage Loading Package, Set Package Defaults, Result Set

The query returns two fields. The data from the query should be saved to the package variables User::RootFilePath and User::EmailMessageFromLine. These variable values will be used later in the package processing.

3. Delete Previously Staged Data

The next task is to clear out any previously staged data for the selected reporting period.

General	General
Parameter Mapping	Name Delete Previously Staged Records
Result Set	Description Execute SQL Task
Expressions	
	Options
	TimeOut 0
	CodePage 1252
	Result Set
	ResultSet None
	SQL Statement
	ConnectionType OLE DB
	Connection EDFacts
	SQLSourceType Direct input
	SQLStatement delete from EDFacts_Staging.Unit_StudentDemog
	IsQueryStoredProcedure False
	BypassPrepare True

Screen 5: Stage Loading Package, Delete Previously Staged Data, General Information

This package won't return any data so the ResultSet value is “None”.

In the case of the load Student Demographics package, the query is very simple.

```
delete
from EDFacts_Staging.Unit_StudentDemographics
where schoolYear = ?
```

If the data source is a text or Excel file, then the data are copied into a table in EDFacts_LocalSource. It is that table that needs to be cleared in the query above. We will clear the EDFacts_Staging table as part of the Post Load Processing step.

There is just 1 parameter, the schoolYear which is mapped to User::ReportingPeriod.

General	Variable Name	Direction	Data Type	Parameter Name	Parameter Size
Parameter Mapping	User::ReportingPeriod	Input	VARCHAR	0	9
Result Set					
Expressions					

Screen 6: Stage Loading Package, Delete Previously Staged Data, Parameter Mapping

This task does not return any data, so it is now complete

4. Copy Records to Staging

The “meat” of the Stage Loading packages is carried out in the next two tasks. This Data Flow task is the one that will query the source system, count the records, and write the data to the appropriate table.

Data typically come from 1 of two source types, either a database table accessible to ES3 via an SSIS data connection, or an Excel or text file accessible via a file system directory. The data flow task varies slightly between these two options.

a. Source Data From a Database

When the source data are in a data base table, ES3 uses an OLEDB connection manager to link to the data. Using an SSIS Connection manager means ES3 does not have to create a database link between the EDFacts database and the source database. Such a connection could be a security hole.

The source database does need to be accessible to the SSIS package when it runs from the EDFacts SSIS repository however.

The typically data flow is shown below.

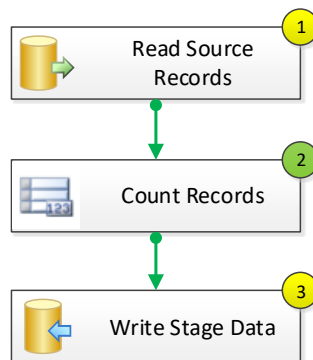
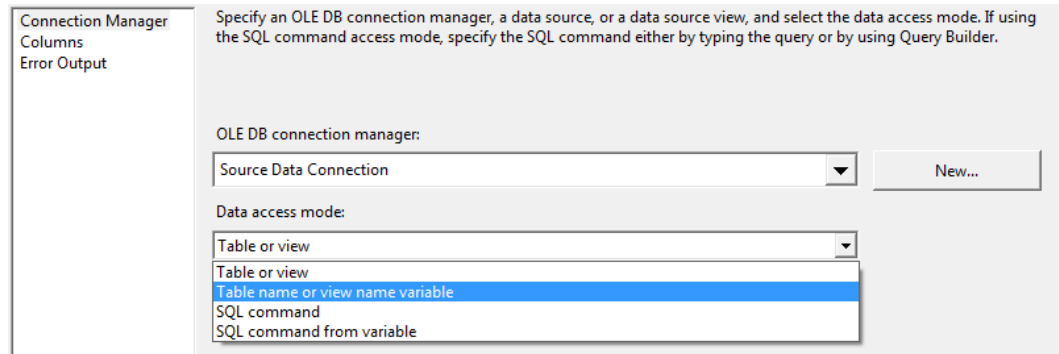


Figure 3: Stage Loading, Database Source Data, Typical Data Flow

The typical flow consists of 1) reading the source data, 2) counting the resulting records, and 3) saving the results. Steps 1 and 3 are by necessity custom for each Stage Loading package; step 2 can remain unchanged.

a 1. Read Source Records

The first step in defining the Read Source Data step is to define the Connection and query to use.



Screen 7: Stage Loading, Source Data from Database, Read Source Records, Connection Manager

Select an existing connection manager or create a New connection using the [New...] button.

There are 4 modes available for data access, as shown in the drop-down list above.

- Table or View – a simple selection of all records in the selected table or view. The name is selected from a drop-down list after this selection is made.
- Table name or view name variable – the name of the table or view to be queried is stored in a package variable. The variable that holds the name is selected from a drop-down list after this selection is made. That means the variable must already exist and contain a valid table or view name.
- SQL Command – after making this selection, a larger text box will appear where a SQL query command can be entered. You have the option of building a query using the Query Builder. The query can contain parameters – enter as “?”s and define on the Parameters... button.
- SQL Command from variable – if you have a string variable that holds the text of a SQL command, you can select this option. The SQL Command from variable option give more flexibility in the way parameters are mapped from package variables. The SQL Command text is limited to 4,000 characters.

The columns tab will show the columns to be returned by the connection defined above. You can deselect some fields if you don't want them all. This step ensures that the system is able to parse your query.

a 2. Count Records

Step 2 is to count the records returned by the query and save the results into the SuccessRecordsCount package variable.

Common Properties	
ComponentClassID	{150E6007-7C6A-4CC3-8FF3-FC73783A972E}
ContactInfo	Row Count;Microsoft Corporation; Microsoft SqlServer v10; (C) Microsoft Corporation;
Description	Counts the rows in a dataset.
ID	391
IdentificationString	component "Row Count" (391)
IsDefaultLocale	True
LocaleID	English (United States)
Name	Row Count
PipelineVersion	0
UsesDispositions	False
ValidateExternalMetadata	True
Version	0

Custom Properties	
VariableName	User::SuccessRecordCount

Screen 8: Stage Loading, Source Data from Database, Count Records

The value stored in this variable will be used later to determine if the package processed successfully. We assume if SuccessRecordCount is non-zero, that we processed successfully.

a 3. Write Staging Records

Finally we write the records to our staging table.

Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder. For fast-load data access, set the table update options.

OLE DB connection manager: EDFacts New...

Data access mode: Table or view - fast load

Name of the table or the view: [EDFacts_Staging].[Unit_StudentDemographics] New...

☐ Keep identity ☒ Table lock

☐ Keep nulls ☒ Check constraints

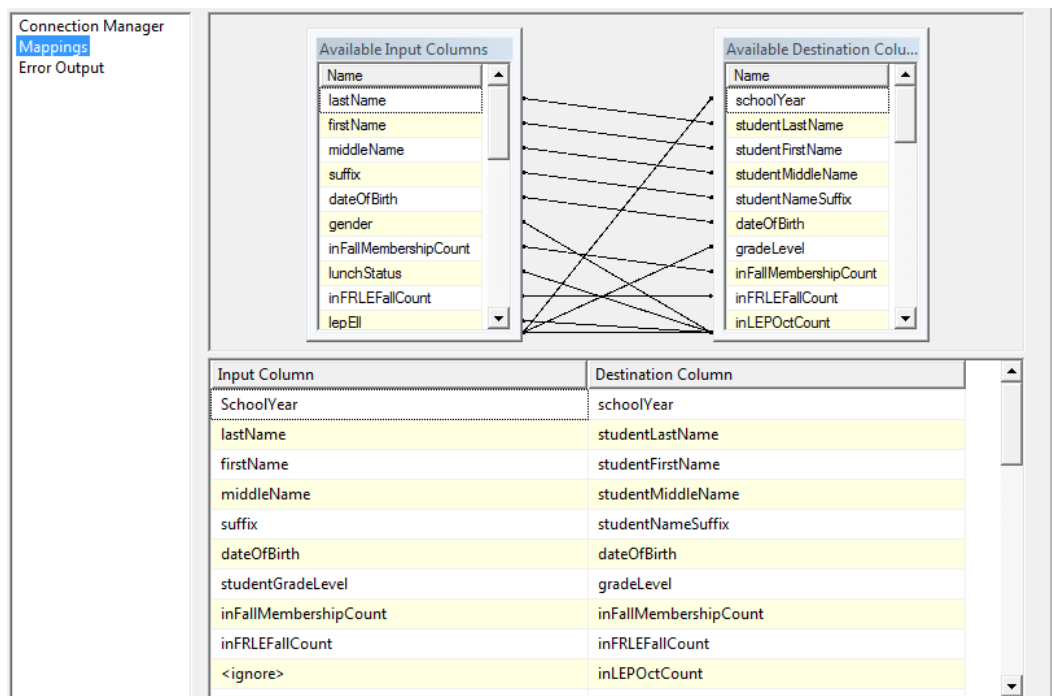
Rows per batch:

Maximum insert commit size: 2147483647

Screen 9: Stage Loading, Source Data from Database, Write Staging Records, Connection Manager

Use an OLEDB destination task and select the EDFacts Connection manager, then select the table into which ES3 needs to copy data.

Then select the Mappings tab to map which input field gets loaded into which destination field.



Screen 10: Stage Loading, Source Data from Database, Write Staging Records, Column Mapping

Either drag a field OR pick the appropriate field from the drop downs in the Input Column section of the bottom half of the screen.

b. Source Data From an Excel File

If the source data is an Excel file or a text file, the typical data flow task is a little different.

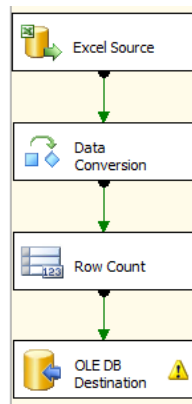
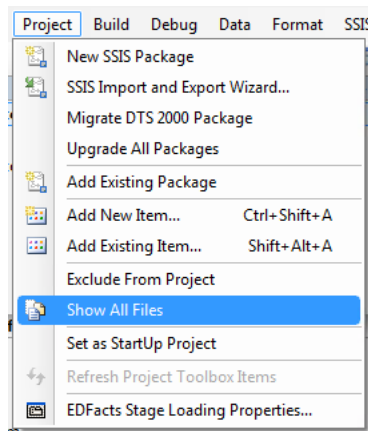


Figure 4: Stage Loading, Excel Source Data, Typical Data Flow

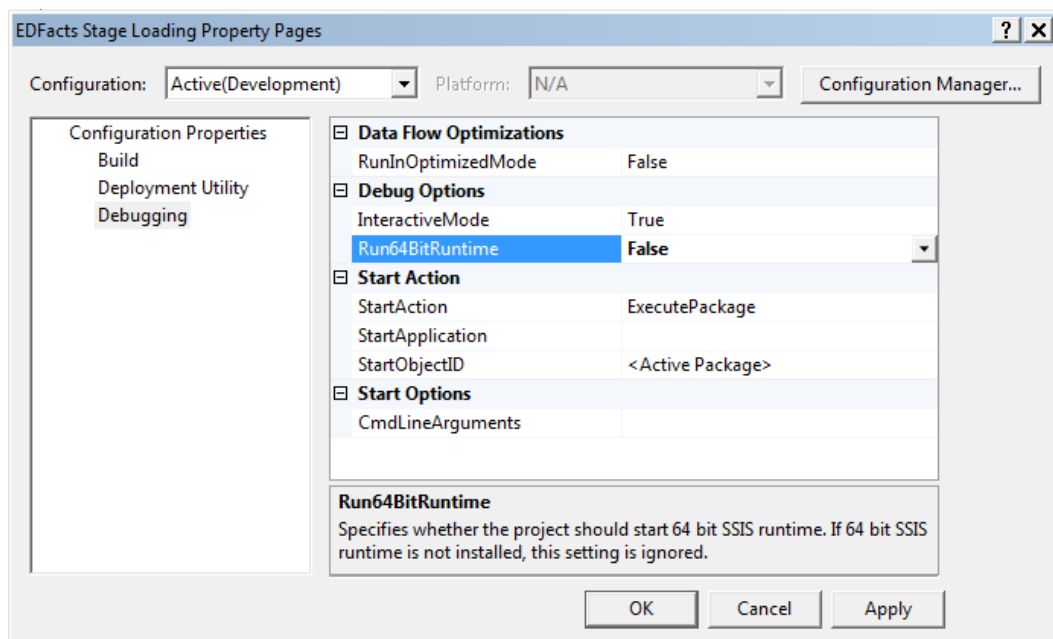
Excel data are read by default using unicode characters. We need to add a Data Conversion task to prepare the data for loading into our database tables.

Also, if the source data come from Excel, we to make sure the SSIS package does not run in 64 bit mode – the Excel connection manager needs to run 32-bit.



Screen 11: Stage Loading, Excel Data Source, Project Properties

Select the project properties for the bottom of the Project menu.



Using the Debugging option on the left, then set Run64BitRuntime to False.

b 1. Excel Source Connection

The first step is to connect to the Excel workbook and the sheet from which to read the source data.

Screen 12: Stage Loading, Excel Source Data, Excel Source Connection, Connection Manager

Use the [New...] button to select the Excel workbook and then the Name of the Excel Sheet drop-down to select the spreadsheet.

External Column	Output Column
School Year	School Year
State Name	State Name
ID 559 FIPS State Code	ID 559 FIPS State Code
ID 570 State Agency Number	ID 570 State Agency Number
ID 4 LEA Identifier (State)	ID 4 LEA Identifier (State)
ID 1 LEA Identifier (NCES)	ID 1 LEA Identifier (NCES)
ID 7 LEA Name	ID 7 LEA Name
ID 669 Out of State Indicator	ID 669 Out of State Indicator
ID 453 LEA Educational Agency Type	ID 453 LEA Educational Agency Type
ID 11 LEA Web Site Address	ID 11 LEA Web Site Address

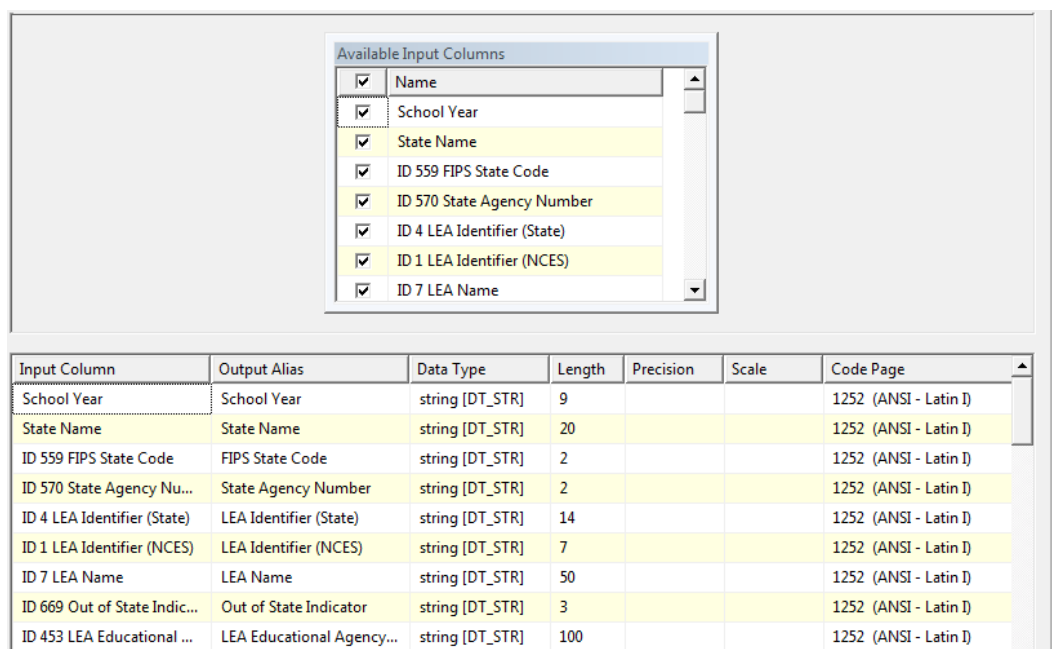
Screen 13: Stage Loading, Excel Source Data, Excel Source Connection, Columns

This screen provides verification that the spreadsheet can be read by the ES3 SSIS process.

b 2. Data Conversion

Excel data is read by SSIS using Unicode code pages. These data cannot be used to load the database tables this way, we need to translate the coding to a non-Unicode character set.

When the Data Conversion task is opened, it will display a list of the fields from the Excel file.



The screenshot shows a 'Available Input Columns' dialog box with a list of columns and checkboxes. Below it is a table with columns: Input Column, Output Alias, Data Type, Length, Precision, Scale, and Code Page.

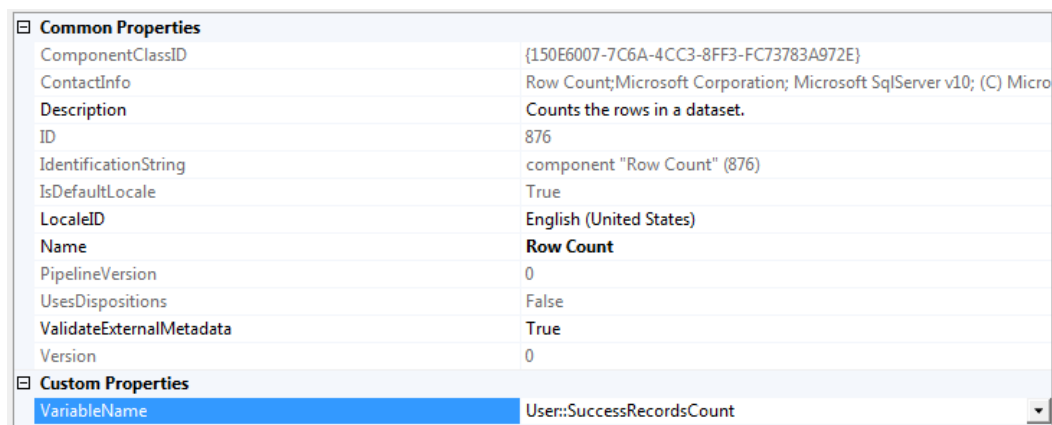
Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
School Year	School Year	string [DT_STR]	9			1252 (ANSI - Latin I)
State Name	State Name	string [DT_STR]	20			1252 (ANSI - Latin I)
ID 559 FIPS State Code	FIPS State Code	string [DT_STR]	2			1252 (ANSI - Latin I)
ID 570 State Agency Nu...	State Agency Number	string [DT_STR]	2			1252 (ANSI - Latin I)
ID 4 LEA Identifier (State)	LEA Identifier (State)	string [DT_STR]	14			1252 (ANSI - Latin I)
ID 1 LEA Identifier (NCES)	LEA Identifier (NCES)	string [DT_STR]	7			1252 (ANSI - Latin I)
ID 7 LEA Name	LEA Name	string [DT_STR]	50			1252 (ANSI - Latin I)
ID 669 Out of State Indic...	Out of State Indicator	string [DT_STR]	3			1252 (ANSI - Latin I)
ID 453 LEA Educational ...	LEA Educational Agency...	string [DT_STR]	100			1252 (ANSI - Latin I)

Screen 14: Stage Loading, Excel Source Data, Data Conversion

For DT_WSTR fields, change the data type to DT_STR and set the Code Page to 1252 (ANSI - Latin 1). By default the output alias will be "Copy of <fieldname>". You can leave this, or rename the output alias to something appropriate.

b 3. Count Records

We still count the records returned by the query and save the results into the SuccessRecordsCount package variable.



The screenshot shows a configuration window with two tabs: 'Common Properties' and 'Custom Properties'.

Common Properties	
ComponentClassID	{150E6007-7C6A-4CC3-8FF3-FC73783A972E}
ContactInfo	Row Count;Microsoft Corporation; Microsoft SqlServer v10; (C) Micro
Description	Counts the rows in a dataset.
ID	876
IdentificationString	component "Row Count" (876)
IsDefaultLocale	True
LocaleID	English (United States)
Name	Row Count
PipelineVersion	0
UsesDispositions	False
ValidateExternalMetadata	True
Version	0
Custom Properties	
VariableName	User::SuccessRecordsCount

Screen 15: Stage Loading, Excel Data Source, Count Records

The value stored in this variable will be used later to determine if the package processed successfully. We assume if SuccessRecordCount is non-zero, that we processed successfully.

b 4. Write Staging Records

This data flow ends with the same Write Staging Records task as the Database source flow.

First select the EDFacts connection manager and desired Staging table.

Connection Manager

Mappings

Error Output

Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder. For fast-load data access, set the table update options.

OLE DB connection manager:

EDFacts

New...

Data access mode:

Table or view - fast load

Name of the table or the view:

[EDFacts_LocalSource].[USEDLEADirectoryAndGradesOffered]

New...

☐ Keep identity

☒ Table lock

☐ Keep nulls

☒ Check constraints

Rows per batch:

Maximum insert commit size:

2147483647

Screen 16: Stage Loading, Excel Data Source, Write Stage Records, Connection Manager

Select the Mappings tab to define which of the fields get mapped into the destination table.

Connection Manager

Mappings

Error Output

Available Input Columns

Name

Data Conversion.LEA Grad...

Data Conversion.LEA Grad...

Data Conversion.LEA Grad...

Data Conversion.LEA Grad...

Data Conversion.LEA Grad...

Data Conversion.LEA Grad...

Data Conversion.LEA Grad...

Data Conversion.LEA No G...

Data Conversion.LEA Grad...

Available Destination C...

Name

serves08

serves09

serves10

serves11

serves12

serves13

servesUG

servesNoGrade

servesAE

webAddress

Input Column	Destination Column
Data Conversion.School Year	schoolYear
Data Conversion.State Name	stateName
FIPS State Code	FIPStateCode
State Agency Number	stateAgencyNumber
LEA Identifier (State)	stateLEAIdentifier
LEA Identifier (NCES)	NCESLEAIdentifier
LEA Name	LEAName
Out of State Indicator	outOfStateIndicator
Telephone - LEA	phoneFormatted
LEA City Location	locationCity

Screen 17: Stage Loading, Excel Source Data, Write Stage Data, Mappings

Either drag a field on the left to a field on the right in the upper half of the window OR pick the appropriate field from the drop downs in the Input Column section of the bottom half of the screen.

Be sure to select the fields on the left from the data conversion task as appropriate. That is, if you converted a string field in the data conversion task, both the pre conversion and post conversion fields will be in the pick lists.

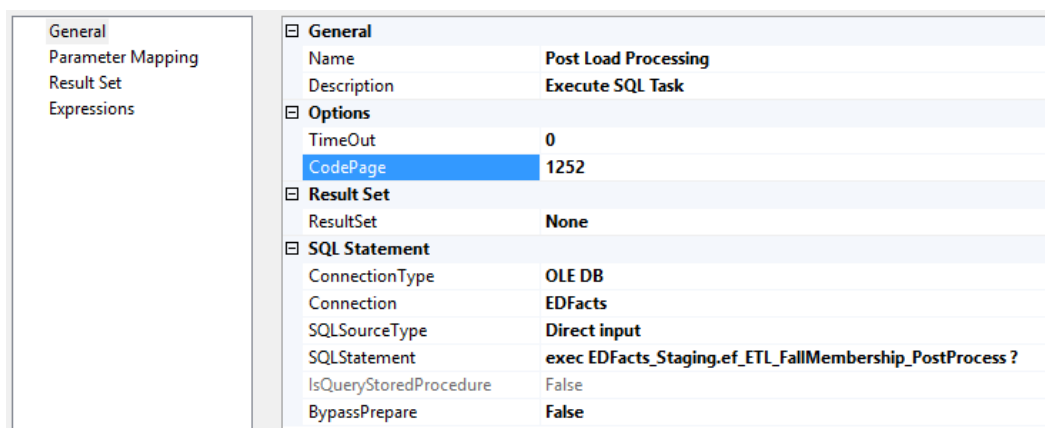
5. Post Load Processing

Whether the source data came from an Excel spreadsheet, a text file, or a database source, there often is post initial load processing and clean-up required to get the data ready for EDFacts. This is the step that stores the states “business rules” for how the EDFacts files should be built.

ES3 uses this step to set the `includeInXXX` flags, get organization names from state identifiers, etc. This is also the step where, if state specific fields were “tacked on” to the staging table, those values are used to populate the core fields that the Submission Loading process uses.

If the source data came from Excel or a text file (that is, if the step above just loaded the `EDFacts_LocalSource` table), then this step clears the selected school year from the `EDFacts_Staging` table and copies the data from `EDFacts_LocalSource` to `EDFacts_Staging`.

This step is typically handled with a stored procedure in the `EDFacts_Staging` schema. The procedure naming convention is `ef_ETL_<Name of the Data Being Processed>_PostProcessing`



Screen 18: Stage Loading, Post Load Processing, General Information

In this case, we have a single stored procedure call as the SQL Statement:

```
exec EDFacts_Staging.ef_ETL_FallMembership_PostProcessing ?
```

The question mark on the end indicates the procedure takes a single parameter, mapped to the `User::ReportingPeriod` variable on the Parameter Mapping step.

General	Variable Name	Direction	Data Type	Parameter Name	Parameter Size
Parameter Mapping	User::ReportingPeriod	Input	VARCHAR	0	9
Result Set					
Expressions					

Screen 19: Stage Loading, Post Load Processing, Parameter Mapping

The stored procedure is custom for each state and Stage Loading package.

6. Build Email

Finally, we need to build the email message. If the successRecordCount is greater than 1, we assume things went well. In this case, we use the SuccessEmailToLine and SuccessEmailCCLine. If not, then we use the ErrorEmailToLine and ErrorEmailCCLine.

```
public void Main()
{
    string strPackageName;
    string strMsgText;

    //Count processing
    int Successes = (int)Dts.Variables["User::SuccessRecordCount"].Value;

    //If no Successes, set warning
    if (Successes == 0)
    {
        //Set result type to warning
        //Get warning message for log
        Dts.Variables["User::EmailMessageSubject"].Value
            = "Warning: No records were copied to Staging";
        Dts.Variables["User::EmailMessageToLine"].Value =
Dts.Variables["User::ErrorEmailToLine"].Value;
        Dts.Variables["User::EmailMessageCCLine"].Value =
Dts.Variables["User::ErrorEmailCCLine"].Value;
    }
    else
    {
        //Get Success message for log
        Dts.Variables["User::EmailMessageSubject"].Value
            = "Success: "
            + Successes.ToString()
            + " records were copied to Staging";

        Dts.Variables["User::EmailMessageToLine"].Value =
Dts.Variables["User::SuccessEmailToLine"].Value;
        Dts.Variables["User::EmailMessageCCLine"].Value =
Dts.Variables["User::SuccessEmailCCLine"].Value;
    }

    strPackageName = (string)Dts.Variables["System::PackageName"].Value;

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

    strMsgText = "The processing of package " + strPackageName;
    strMsgText += " completed for the ";
    strMsgText += Dts.Variables["User::ReportingPeriod"].Value;
    strMsgText += " reporting period.  \n\n";

    strMsgText += Successes.ToString();
    strMsgText += " records were written to the ";
}
```

```

strMsgText += Dts.Variables["StagingTableName"].Value;
strMsgText += " Table.";

Dts.Variables["User::EmailMessageBody"].Value = (object)strMsgText;

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

```

Read Only Variables

- User::ErrorEmailCCLine
- User::ErrorEmailToLine
- System::PackageName
- User::ReportingPeriod
- User::SuccessEmailCCLine
- User::SuccessEmailToLine
- User::SuccessRecordCount

ReadWrite Variables

- User::EmailMessageBody
- User::EmailMessageCCLine
- User::EmailMessageSubject
- User::EmailMessageToLine

No edits are required from the template. However, depending on the data being staged, sometimes additional information is useful and the email may be adjusted as needed.

7. 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.

8. 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:

General	Variable Name	Direction	Data Type	Parameter Name	Parameter Size
Parameter Mapping	User::EmailMessageSubject	Input	NVARCHAR	0	-1
Result Set	User::EmailMessageToLine	Input	NVARCHAR	1	-1
Expressions	User::EmailMessageCCLine	Input	NVARCHAR	2	-1
	User::EmailMessageBody	Input	NVARCHAR	3	-1
	User::EmailAttachments	Input	NVARCHAR	4	-1

Screen 20: Stage Loading, Update Email Log, Parameter Mapping

No edits are required from the template.

9. Send Notification

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

General	Mail
Mail	
Expressions	

Mail	
SmtConnection	MailServer
From	EDFacts@edu.state.gov
To	EDFacts_God@edu.state.gov
Cc	
BCc	
Subject	Process Failed
MessageSourceType	Variable
MessageSource	User::EmailMessageBody
Priority	Normal
Attachments	

Screen 21: Stage Loading, Send Notification, Mail Settings

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

General	Misc
Mail	
Expressions	

Misc	
Expressions	
CCLine	@[User::EmailMessageCCLine]
FileAttachments	@[User::EmailAttachments]
FromLine	@[User::EmailMessageFromLine]
Subject	@[User::EmailMessageSubject]
ToLine	@[User::EmailMessageToLine]

No edits are required from the template.

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 three columns at the start of the table for the report level, school year (reporting period), and a categorySortOrder to group the records by category set or subtotal number.

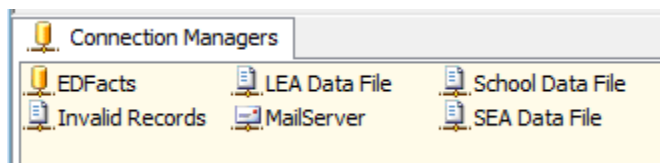
In a few instances, the table structure is different for the different levels, directory for instance. In that case, the level will be appended to the end of the table names: for example S092_SCH or S029_LEA.

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 Submission Package Connection Managers

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.



Screen 23: Submission Loading, Generic Connection Managers

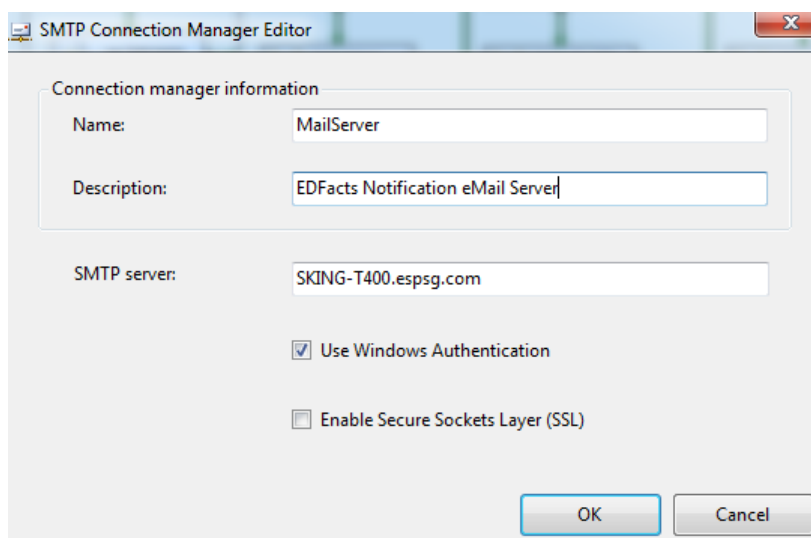
EDFacts

This is an OLEDB connection to the EDFacts SQL Server catalog. This catalog has EDFacts_Admin, EDFacts_Compare, EDFacts_LocalSource, EDFacts_Staging, EDFacts_Submission, and EDFacts_Validation schemas. These 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.



Screen 24: Submission Loading, Connections, Mail Server

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

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

The other two settings in the Core Connections package configuration are EnableSsl and UseWindowsAuthentication.

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 RootFileDirectory value comes from the StateConfig table's storageDirectoryRootPath field value for the selected reporting period. The <FileSubDirectory> value comes from EDFacts_Admin.SubmissionFileCharacteristic table. The FileNameSEA value 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 RootFileDirectory value comes from the StateConfig table's storageDirectoryRootPath field value for the selected reporting period. The <FileSubDirectory> value comes from EDFacts_Admin.SubmissionFileCharacteristic table. The FileNameLEA value 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 RootFileDirectory value comes from the StateConfig table's storageDirectoryRootPath field value for the selected reporting period. The <FileSubDirectory> value comes from EDFacts_Admin.SubmissionFileCharacteristic table. The FileNameSchool value 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 Submission Package ETL 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
CreateZipsAndAttach	Set in [Set Package Defaults], Default to "Yes"	Create Email Text tasks and toggles Zip file tasks
CsvFileNameLEA	Set in [Get the Filename to Use for LEA]	[Write LEA Header Rec] and [Write LEA Data Records]
CsvFileNameSchool	Set in [Get the Filename to Use for School]	[Write School Header Rec] and [Write School Data Records]
CsvFileNameSEA	Set in [Get the Filename to Use for SEA]	[Write SEA Header Rec] and [Write SEA Data Records]
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 when the Email is created either as ErrorMessageCCLine or SuccessEmailCCLine	send notification
EmailMessageFromLine	Set in Set Package Defaults. Read from StateConfig	send notification
EmailMessageSubject	Default value of "<package name> Failed" is set in the package. Value gets updated in [Create Email Message Text] or [Create Success Email Text] tasks	send notification
EmailMessageToLine	Set when the Email is created either as ErrorMessageToLine or SuccessEmailToLine	send notification
ErrorEmailCCLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Email Message Text
ErrorEmailToLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Email Message Text
FileSubDirectory		
InvalidRecordsFileName		

Variable	Value Source	Used By
ReportingPeriod		
RootFilePath	read in from the state config table in the Set Filepath routine	send notification
SubmissionFileNumber		
SuccessEmailCCLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Success Email Text
SuccessEmailToLine	Default values set in the package but exposed in the "Package Specific" package configuration. Editable at runtime	Create Success Email Text
SuccessRecordsCount		
ZipFileNameLEA		
ZipFileNameSchool		
ZipFileNameSEA		
ZipFilePassword		
ZipInvalidRecordsFileName		

Generic Submission Package ETL Flow Design

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

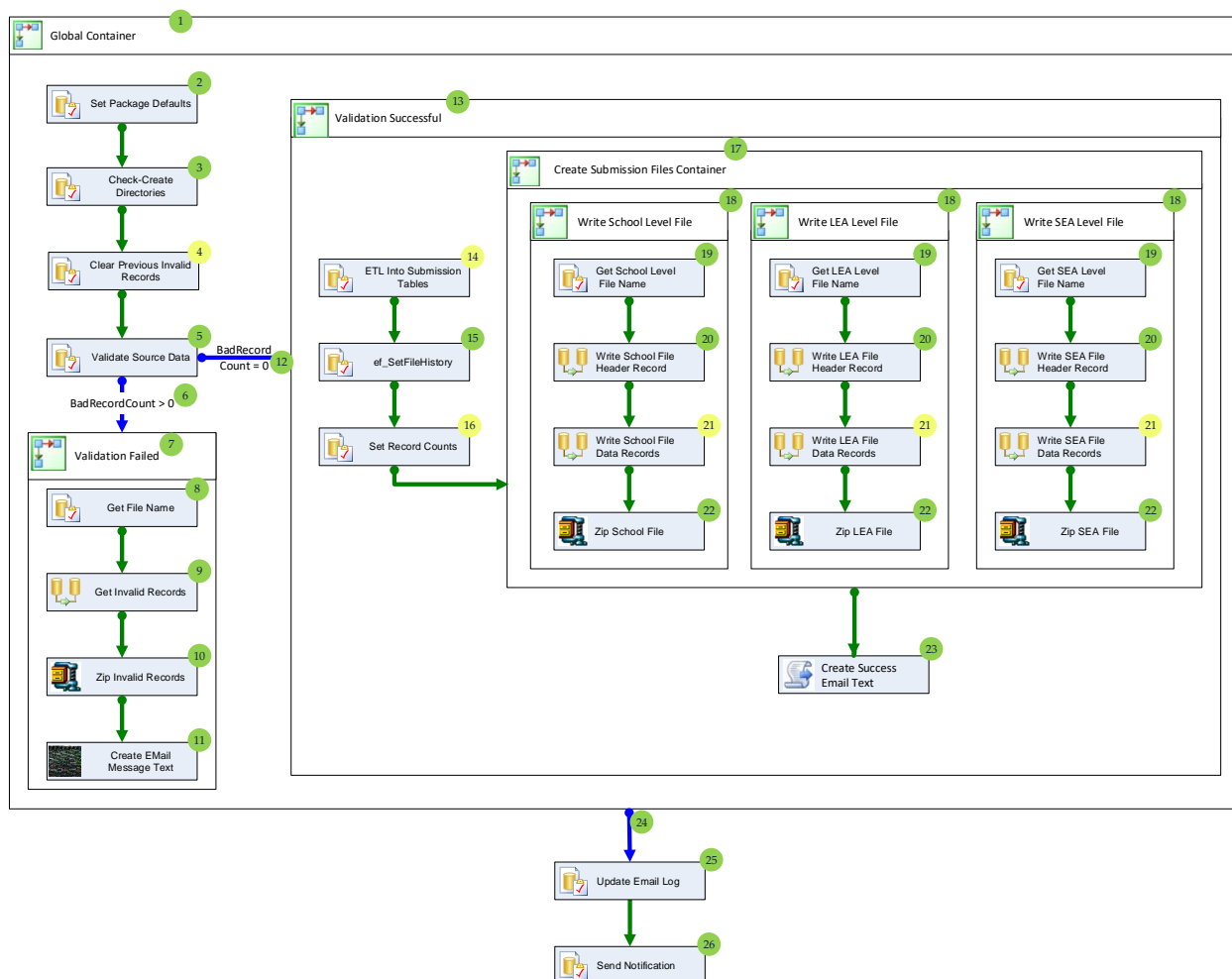


Figure 5: Submission Loading Generic Package Control Flow

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:

- 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)

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
- Invalid Records File Name
- Email Message From: line

General	General
Parameter Mapping	Name Set Package Defaults
Result Set	Description Execute SQL Task
Expressions	Options
	TimeOut 0
	CodePage 1252
	Result Set
	ResultSet Single row
	SQL Statement
	ConnectionType OLE DB
	Connection EDFacts
	SQLSourceType Direct input
	SQLStatement select top 1 cfg.storageDirectoryRoo
	IsQueryStoredProcedure False
	BypassPrepare True

Screen 25: Submission Loading, Set Package Defaults, General Information

The query is shown below

```
select top 1
    cfg.storageDirectoryRootPath,
    fc.fileSubdirectory,
    fc.invalidRecordsFileName,
    cfg.EmailMessageFromLine
from EDFacts_Admin.SubmissionFileCharacteristic fc
join EDFacts_Admin.StateConfig cfg
    on (fc.reportingPeriod = cfg.reportingPeriod)
where fc.reportingPeriod = ?
    and fc.specificationNumber = ?
```

There are two parameters for the query: the submission file number and the reporting period. The submissionFileNumber is a variable that is set at design time. The reportingPeriod is a variable that the user will be able to change via a configuration file.

General	Variable Name	Direction	Data Type	Parameter Name	Parameter Size
Parameter Mapping	User::SubmissionFileNumber	Input	VARCHAR	1	-1
Result Set	User::ReportingPeriod	Input	VARCHAR	0	9
Expressions					

Screen 26: Submission Loading, Set Package Defaults, Parameter Mapping

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

General Parameter Mapping Result Set Expressions	<table> <tr> <th>Result Name</th><th>Variable Name</th></tr> <tr> <td>0</td><td>User::RootFilePath</td></tr> <tr> <td>1</td><td>User::FileSubDirectory</td></tr> <tr> <td>2</td><td>User::InvalidRecordsFileName</td></tr> <tr> <td>3</td><td>User::EmailMessageFromLine</td></tr> </table>	Result Name	Variable Name	0	User::RootFilePath	1	User::FileSubDirectory	2	User::InvalidRecordsFileName	3	User::EmailMessageFromLine
Result Name	Variable Name										
0	User::RootFilePath										
1	User::FileSubDirectory										
2	User::InvalidRecordsFileName										
3	User::EmailMessageFromLine										

Screen 27: Submission Loading, Set Package Defaults, Result Set

3. Check-Create Directories

This script task verifies that the root directory (RootFilePath variable) and the package sub-directory (FileSubDirectory variable) exist on the server. If the root directory cannot be found, then the email message is created stating this directory is missing and the package is failed.

If the specific package subdirectory is missing, then the directory is created within the root directory.

Script

General

Expressions

Script

ScriptLanguage

EntryPoint

ReadOnlyVariables

ReadWriteVariables

Microsoft Visual C# 2008

Main

User::FileSubDirectory, System::PackageName, User::RootFilePath

User::EmailMessageBody, User::EmailMessageSubject

Screen 28: Submission Loading, Check-Create Directories, Script Settings

```
public void Main()
{
    // Specify the directory you want to check.
    string pathRoot = (string)Dts.Variables["User::RootFilePath"].Value;
    string pathSub = (string)Dts.Variables["User::FileSubDirectory"].Value;

    // Determine whether the main directory exists.
    if (Directory.Exists(pathRoot))
    {
        // Main Directory exists
        // Check if subfolder exists. If not, then create the folder.
        string fullPathSub = Path.Combine(pathRoot, pathSub);

        if (!Directory.Exists(fullPathSub))
        {
            DirectoryInfo di = Directory.CreateDirectory(fullPathSub);
        }
    }
    else
    {
        // Main Directory does not exist.
    }
}
```

```
// Create an email message and fail the package immediately
string strMsgText;
string strPackageName = (string)Dts.Variables["System::PackageName"].Value;

strMsgText = "The package '" + strPackageName + "' failed. ";
strMsgText += "The folder '" + pathRoot + "' does not exist.";

Dts.Variables["User::EmailMessageSubject"].Value = strPackageName + " failed";
Dts.Variables["User::EmailMessageBody"].Value = (object)strMsgText;

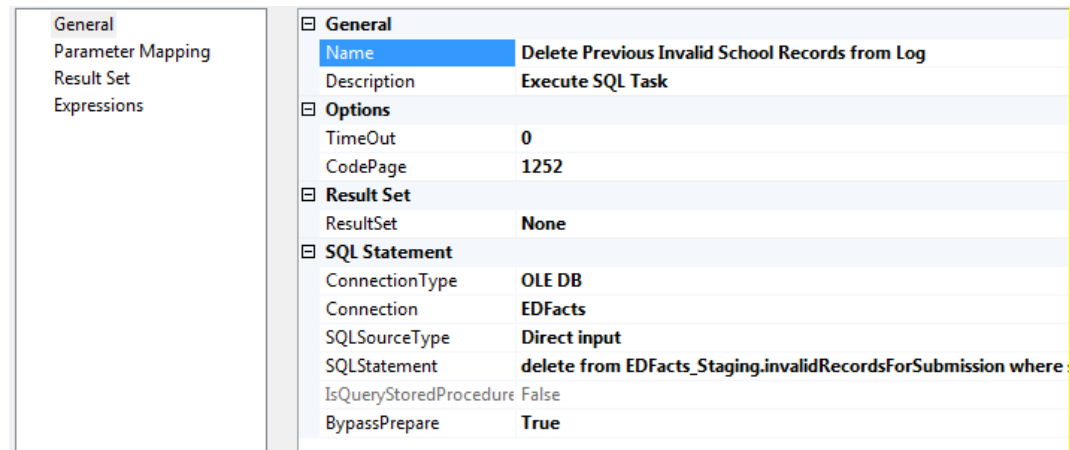
Dts.TaskResult = (int)ScriptResults.Failure;
return;

}

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

4. 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.

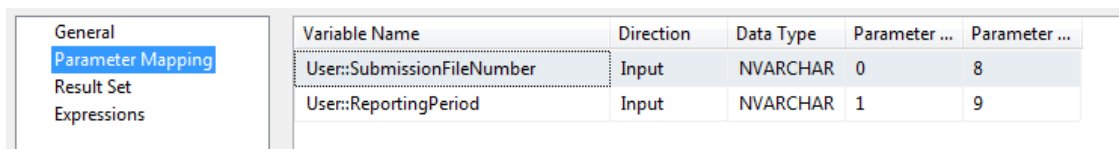


Screen 29: Submission Loading, Clear Previous Invalid Records, General Information

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.



Screen 30: Submission Loading, Clear Previous Invalid Records, Parameter Mapping

No edits are required from the template.

5. Validate the Source Data

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

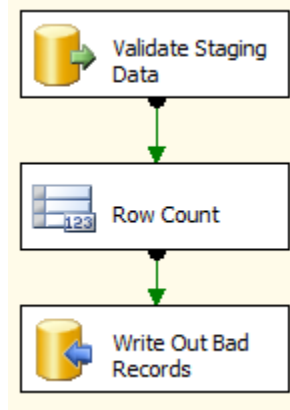


Figure 6: Submission Loading, Validate Source Data, Data Flow

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 looks like:

```

create procedure [EDFacts_Staging].[ef_StageValidation_S052]
    @SchoolYear as varchar (9)
as
/**
 * Runs validation against the N052 source (Unit_studentDemographics) and
 * SELECTS any bad records. In the SSIS package, if record count > 0 redirect
 * the process to report the error records and stop the EDFacts file creation
 *
 * @author      : Steven King, ESP Solutions Group
 * @version     : 1.0 28-Oct-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      : 'Select' bad records for saving into the invalid records
 *                  for submission table. The structure of the Invalid
 *                  records table is:
 *                  SubmissionFile  varchar(8)  e.g. @SpecificationFileNumber
 *                  reportingPeriod  varchar(9)  '2010-2011'
 *                  keyFieldValue    varchar(50)  <student ID> or other
 *                  unique ID for the
 *                  invalid record
 *                  errorMessage     varchar(150) <error message text>
 * Revision History
 * -----
 *
 */
begin
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
  
```



```

set nocount on;
declare @FileSpecificationNumber varchar (4) = 'S052'
declare @ProblemRecords table (
    keyFieldValue varchar (50),
    errorMessage varchar (150)
)
declare @CodeSetExists int
declare @CodeSetName varchar (50)

-----
--      C O D E   S E T S
-----
--      Code Sets needed in StateCodeTranslation table
--      :      Grade Level (Membership)
--      :      Sex
--      :      Race Ethnicity
-----
-- region  G r a d e   L e v e l   ( M e m b e r s h i p )
--
set @CodeSetName = 'Grade Level (Membership)'

-- check code set defined in code set Translation table
set @CodeSetExists =
    ( select count (*)
      from EDFacts_Admin.StateCodeTranslation
      where codeSetName = @CodeSetName
            and reportingPeriod = @SchoolYear)

-- Report error if not
insert into @ProblemRecords (
    keyFieldValue,
    errorMessage
)
select @CodeSetName as keyFieldValue,
       'Code Set Missing from StateCodeTranslation table' as errorMessage
where @CodeSetExists = 0

-- Check staging values are in the Code set, but only if code set exists
if @CodeSetExists >= 1
begin
    insert into
        @ProblemRecords (keyFieldValue, errorMessage)
    select 'State Student ID: '
        + s.stateStudentIdentifier
        as KeyFieldValue,                                     -- Set KeyValue
        'Invalid option for ['
        + @CodeSetName
        + '] of ['
        + isnull(s.gradeLevel, '')
        + ']'
        as errorMessage                                       -- update error message
    from EDFacts_Staging.Unit_Studentdemographics s
    left join
        EDFacts_Admin.StateCodeTranslation t
        on      t.reportingPeriod = @SchoolYear
              and t.codeSetName = @CodeSetName
              and s.gradeLevel = t.stateCode
    where t.edfactsCode is null
          and s.schoolYear = @SchoolYear
end
-- end region

-----
-- region  R a c e   E t h n I c I t y
-----
set @CodeSetName = 'Race Ethnicity'

```

```

-- check code set defined in code set Translation table
set @CodeSetExists =
    ( select count (*)
      from EDFacts_Admin.StateCodeTranslation
      where codeSetName = @CodeSetName
        and reportingPeriod = @SchoolYear)

-- Report error if not
insert into @ProblemRecords (
    keyFieldValue,
    errorMessage
)
select @CodeSetName as keyFieldValue,
    'Code Set Missing from StateCodeTranslation table' as errorMessage
where @CodeSetExists = 0

-- Check staging values are in the Code set, but only if code set exists
if @CodeSetExists >= 1
begin
    insert into
        @ProblemRecords (keyFieldValue, errorMessage)
    select 'State Student ID: '
        + s.stateStudentIdentifier
        as KeyFieldValue,                                -- Set KeyValue
        'Invalid option for ['
        + @CodeSetName
        + '] of ['
        + isnull(s.raceEthnic, '')
        + ']'
        as errorMessage                                  -- update error message
    from EDFacts_Staging.Unit_Studentdemographics s
    left join
        EDFacts_Admin.StateCodeTranslation t
    on
        t.reportingPeriod = @SchoolYear
        and t.codeSetName = @CodeSetName
        and s.raceEthnic = t.stateCode
    where t.edfactsCode is null
        and s.schoolYear = @SchoolYear
    end
-- end region
-----
-- region S e x
--

set @CodeSetName = 'Sex'

-- check code set defined in code set Translation table
set @CodeSetExists =
    ( select count (*)
      from EDFacts_Admin.StateCodeTranslation
      where codeSetName = @CodeSetName
        and reportingPeriod = @SchoolYear)

-- Report error if not
insert into @ProblemRecords (
    keyFieldValue,
    errorMessage
)
select @CodeSetName as keyFieldValue,
    'Code Set Missing from StateCodeTranslation table' as errorMessage
where @CodeSetExists = 0

-- Check staging values are in the Code set, but only if code set exists
if @CodeSetExists >= 1
begin
    insert into
        @ProblemRecords (keyFieldValue, errorMessage)
    select 'State Student ID: '

```

```

+ s.stateStudentIdentifier
  as KeyFieldValue,                                -- Set KeyValue
  'Invalid option for ['
+ @CodeSetName
+ ']' of ['
+ isnull(s.sex, '')
+ ']'
  as errorMessage                                -- update error message
from EDFacts_Staging.Unit_Studentdemographics s
left join
  EDFacts_Admin.StateCodeTranslation t
on    t.reportingPeriod = @SchoolYear
      and t.codeSetName = @CodeSetName
      and s.sex = t.stateCode
where t.edfactsCode is null
      and s.schoolYear = @SchoolYear
end
-- end region

-- Now select the data to be returned
select
  @FileSpecificationNumber as submissionFile,
  @SchoolYear as reportingPeriod,
  keyFieldValue,
  errorMessage
from @ProblemRecords
end

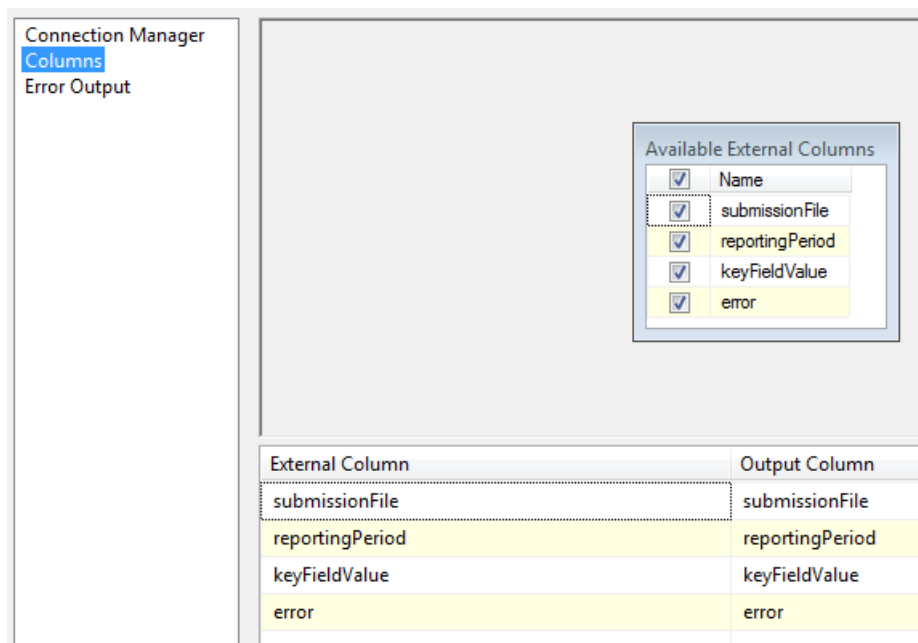
```

The structure of the routine is to create a table variable to hold any details about any erroneous records. There are two fields in the table, one for record identification information (key field) and then the error message.

There are sections to validate each of the code sets needed from the StateCodeTranslation table. The first part of the section verifies that the code set exists, then if it does, are all of the values in the staging table present in the translation table.

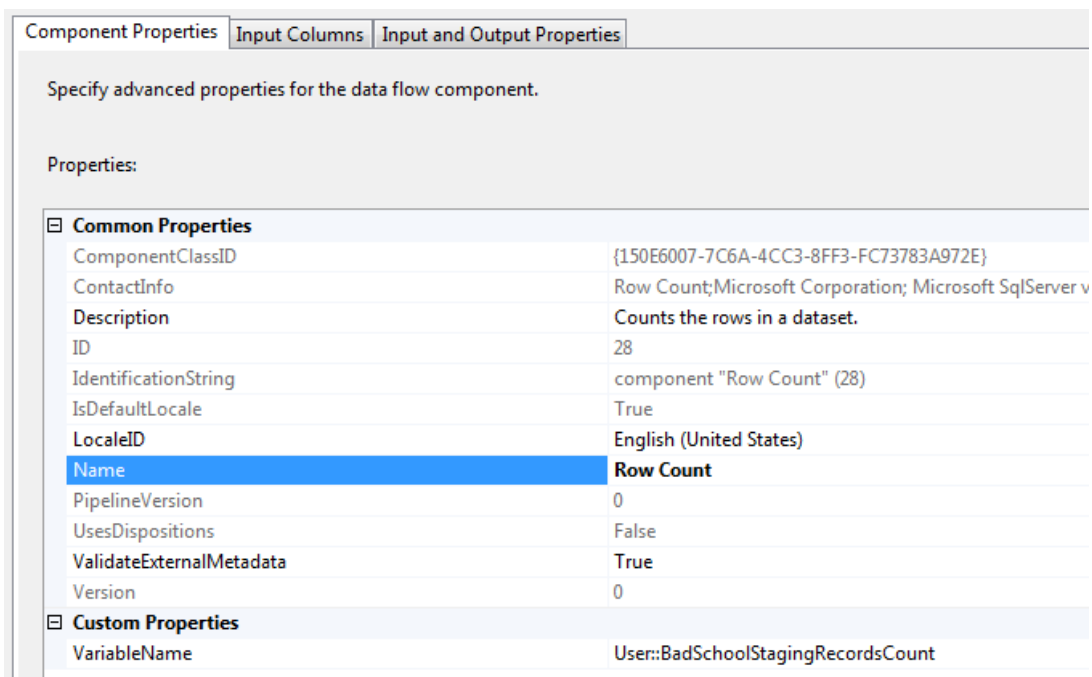
Finally, any results in the table variable are “SELECTed” as the result for the SSIS data flow task. The selected fields are:

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



Screen 31: Submission Loading, Validate Source Data, Columns

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.



Screen 32: Submission Loading, Validate Source Data, Bad Records Count

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

Specify an OLE DB connection manager, a data source, or a data source view, and select the data the SQL command access mode, specify the SQL command either by typing the query or by using fast-load data access, set the table update options.

OLE DB connection manager:
EDFacts

Data access mode:
Table or view - fast load

Name of the table or the view:
[EDFacts_Staging].[InvalidRecordsForSubmission]

Screen 33: Submission Loading, Validate Source Data, Write Invalid Records, Connection Manager

Input Column	Destination Column
submissionFile	submissionFile
reportingPeriod	reportingPeriod
keyFieldValue	keyFieldValue
errorMessage	errorMessage

Screen 34: Submission Loading, Validate Source Data, Write Invalid Records, Field Mapping

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.

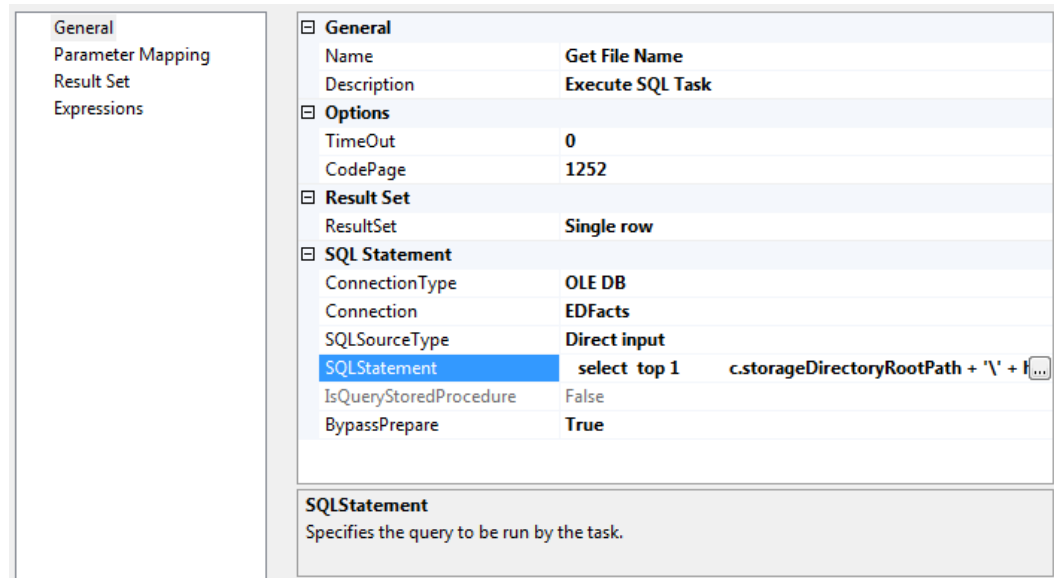
7. Validation Failed Container

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.



General	
Name	Get File Name
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
Result Set	
ResultSet	Single row
SQL Statement	
ConnectionType	OLE DB
Connection	EDFacts
SQLSourceType	Direct input
SQLStatement	select top 1 c.storageDirectoryRootPath + '\'' + I...
IsQueryStoredProcedure	False
BypassPrepare	True
SQLStatement Specifies the query to be run by the task.	

Screen 35: Submission Loading, Get Invalid Records File Name, General Information

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
```

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

General	Variable Name	Direction	Data Type	Parameter Name	Parameter Size
Parameter Mapping	User::ReportingPeriod	Input	VARCHAR	0	9
Result Set	User::SubmissionFileNumber	Input	VARCHAR	1	8
Expressions					

Screen 36: Submission Loading, Get Invalid Records File Name, Parameter Mapping

And it saves its information into three variables

General	Result Name	Variable Name
Parameter Mapping	0	User::InvalidRecordsFileName
Result Set	1	User::ZipInvalidRecordsFileName
Expressions	2	User::ZipFilePassword

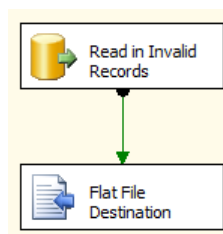
Screen 37: Submission Loading, Get Invalid Records File Name, Result Set

No changes are need for this task from the template.

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



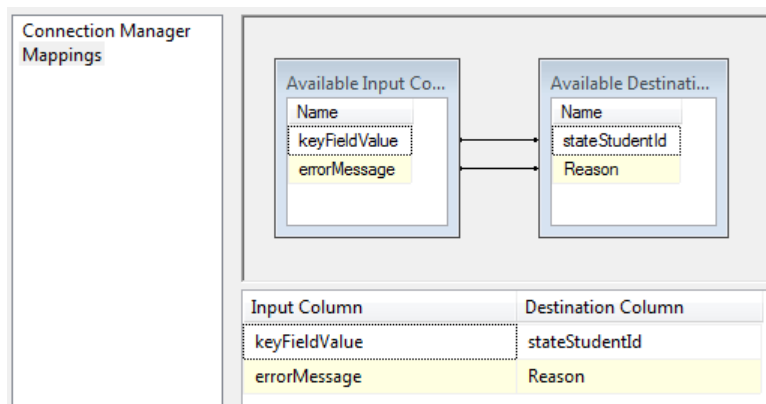
Screen 38: Submission Loading, Get Invalid Records Data Flow

The first takes the current submission file and reporting period to query out the keyRecordValue and ErrorMessage 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.

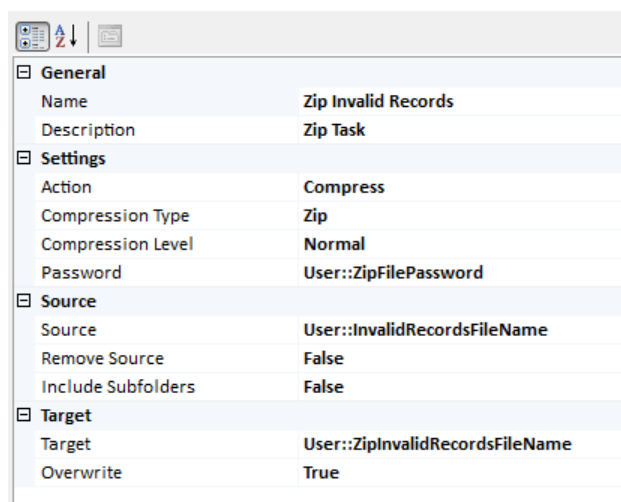


Screen 39: Submission Loading, Get Invalid Records, Mappings

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.



Screen 40: Submission Loading, Zip Invalid Records File

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 ErrorMessageBody user variable.

The ErrorMessageToLine and ErrorMessageCCLine are set by copying the values from ErrorEmailToLine and ErrorEmailCCLine, respectively.

The name of the Invalid Records zip file is stored in the EmailAttachments variable.

Script	
ScriptLanguage	Microsoft Visual C# 2008
EntryPoint	Main
ReadOnlyVariables	User::BadRecordsCount,User::ErrorEmailCCLine,User::ErrorEmailToLine,User::InvalidRecordsFileName,System::PackageName,User::ZipFilePassword,User::ZipInvalidRecordsFileName
ReadWriteVariables	User::EmailAttachments,User::EmailMessageBody,User::EmailMessageCCLine,User::EmailMessageSubject,User::EmailMessageToLine

Screen 41: Submission Loading, Create Error Email Message, General Settings

This is hard to read. The ReadOnly variables are:

- User::BadRecordsCount
- User::ErrorEmailCCLine
- User::ErrorEmailToLine
- User::InvalidRecordsFileName
- System::PackageName
- User::ZipFilePassword
- User::ZipInvalidRecordsFileName

The ReadWrite variables are:

- User::EmailAttachments
- User::EmailMessageBody
- User::EmailMessageCCLine
- User::EmailMessageSubject
- User::EmailMessageToLine

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 strPackageName;
    string strZipFileName;
    string strAttachments;

    strPackageName = (string)Dts.Variables["System::PackageName"].Value;
    strZipFileName = (string)Dts.Variables["User::ZipInvalidRecordsFileName"].Value;
    strWorkFile = (string) Dts.Variables["User::InvalidRecordsFileName"].Value;

    Dts.Variables["User::EmailMessageSubject"].Value =
        strPackageName
        + " validation failed";

    strMsgText = "The package '" + strPackageName;
    strMsgText += "' failed data validation. There ";
    intBadRecordCount = (int) Dts.Variables["User::BadRecordsCount"].Value;

    if (intBadRecordCount != 1)
    {
        strMsgText += "were " + intBadRecordCount.ToString() + " records";
        strMsgText += " that failed.\n";
    }
}
```

```

        if (intBadRecordCount < 20)
        {
            strMsgText += "\nThe record ID's and reasons for the failure ";
            strMsgText += "are listed below:\n\n";
        }
        else
            strMsgText += "\nThe first 20 invalid records are listed below: \n\n";
    }
    else
    {
        strMsgText += "was 1 record the failed.\n";
        strMsgText += "\nThe record ID and reason for the failure are ";
        strMsgText += "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 ";
    strMsgText += " are included 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::EmailMessageToLine"].Value =
        Dts.Variables["User::ErrorEmailToLine"].Value;
    Dts.Variables["User::EmailMessageCCLine"].Value =
        Dts.Variables["User::ErrorEmailCCLine"].Value;

    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

General	Parameter Mapping	Result Set	Expressions
<div> <div>General</div> <div> Nameef_ETL_S052 DescriptionExecute SQL Task </div> </div> <div> <div>Options</div> <div> TimeOut0 CodePage1252 </div> </div> <div> <div>Result Set</div> <div> ResultSetNone </div> </div> <div> <div>SQL Statement</div> <div> ConnectionTypeOLE DB ConnectionEDFacts SQLSourceTypeDirect input SQLStatementexec EDFacts_Submission.ef_ETL_S052 ? IsQueryStoredProcedureFalse BypassPrepareTrue </div> </div>			

Screen 42: Submission Loading, Conduct the ETL, General Information

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

General	Parameter Mapping	Result Set	Expressions										
<table border="1"> <thead> <tr> <th>Variable Name</th> <th>Direction</th> <th>Data Type</th> <th>Parameter Name</th> <th>Parameter Size</th> </tr> </thead> <tbody> <tr> <td>User::ReportingPeriod</td> <td>Input</td> <td>VARCHAR</td> <td>0</td> <td>9</td> </tr> </tbody> </table>				Variable Name	Direction	Data Type	Parameter Name	Parameter Size	User::ReportingPeriod	Input	VARCHAR	0	9
Variable Name	Direction	Data Type	Parameter Name	Parameter Size									
User::ReportingPeriod	Input	VARCHAR	0	9									

Screen 43: Submission Loading, Conduct the ETL, Parameter Mapping

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.

General	Parameter Mapping	Result Set	Expressions
<div> <div>General</div> <div> NameSet File History DescriptionExecute SQL Task </div> </div> <div> <div>Options</div> <div> TimeOut0 CodePage1252 </div> </div> <div> <div>Result Set</div> <div> ResultSetNone </div> </div> <div> <div>SQL Statement</div> <div> ConnectionTypeOLE DB ConnectionEDFacts SQLSourceTypeDirect input SQLStatementexec EDFacts_Admin.ef_ETL_SetFileHistory ?,?, 'Y', 'Y', 'Y' IsQueryStoredProcedureFalse BypassPrepareTrue </div> </div>			

Screen 44: Submission Loading, Set File History, General Information

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.

General	Variable Name	Direction	Data Type	Parameter Name	Parameter Size
Parameter Mapping	User::SubmissionFileNumber	Input	VARCHAR	0	8
Result Set	User::ReportingPeriod	Input	VARCHAR	1	9
Expressions					

Screen 45: Submission Loading, Set File History, Parameter Mapping

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
      where reportLevel = 'SCH' and schoolYear = @ReportingPeriod)
where specificationNumber = 'S052'
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
      where reportLevel = 'LEA' and schoolYear = @ReportingPeriod)
where specificationNumber = 'S052'
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
      where reportLevel = 'SEA' and schoolYear = @ReportingPeriod)
where specificationNumber = 'S052'
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

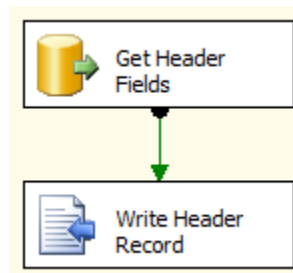
These Execute SQL Task processes get the file name and file path from the SubmissionFileHistory table. The values were set as part of the Set File History step #15.

```
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:



Screen 46: Submission Loading, Write File Header, Data Flow

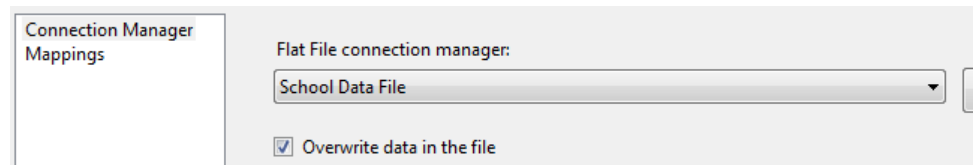
The Get Header Fields OLEBD Data Source task has a query that reads the appropriate information out of the SubmissionFileCharacteristics table and the SubmissionFileHistory table.

```
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.

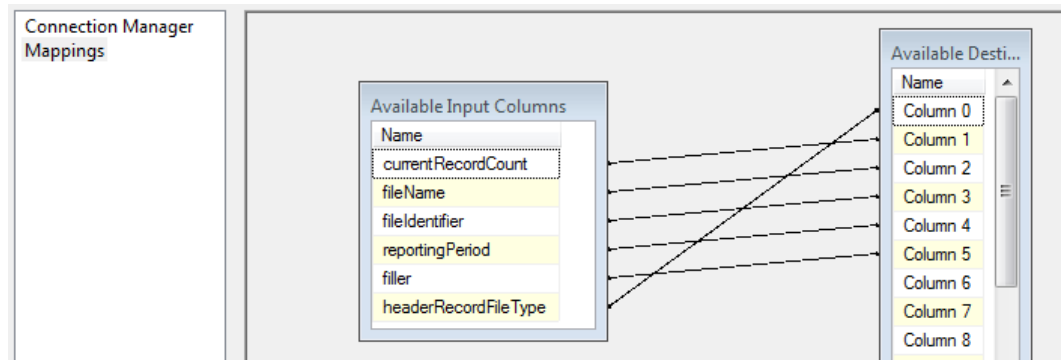
The Write Header Record simply writes the selected data to the first six columns in the appropriate data file connection.

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



Screen 47: Submission Loading, Write File Header, Connection Manager

Map the columns as required by the file specification.



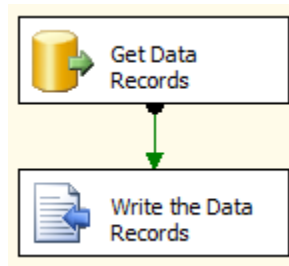
Screen 48: Submission Loading, Write File Header, Mappings

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.



Screen 49: Submission Loading, Write File Data Records, Data Flow

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
            totalIndicator,
            categorySortOrder,
            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 = ?
  
```

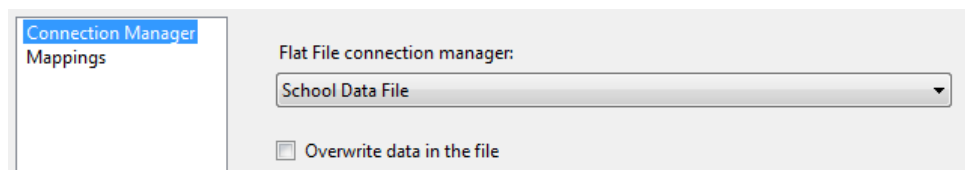
The first field in the query gets the unique line number for each record in the final file. The totalIndicator and categorySortOrder fields in the Order By clause

make sure the records are grouped alphabetically by the category sets first, then numerically by the subtotals, and finally the Total, if applicable.

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

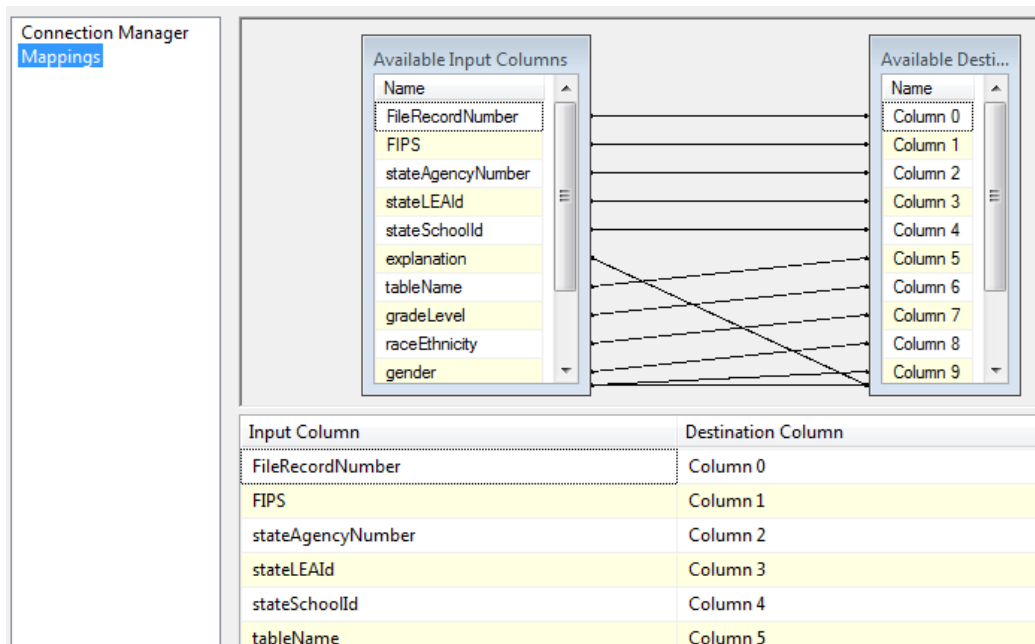
The reportLevel value in the WHERE clause needs to be set for each of the appropriate reporting levels.

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.



Screen 50: Submission Loading, Write File Data Records, Connection Manager

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



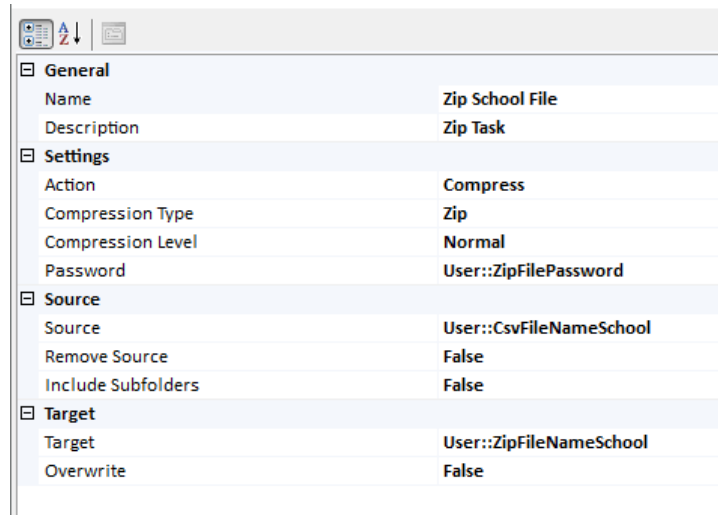
Screen 51: Submission Loading, Write File Data Records, Mappings

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

SubmissionFileCharacteristic table, then that password is applied to the zipped file.

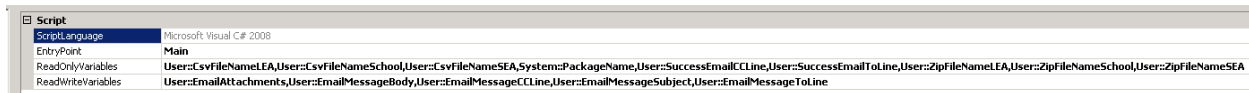


Screen 52: Submission Loading, Zip Submission Files

No edits are required from the Template.

23. Create Success Email Message Text

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



Screen 53: Submission Loading, Create Success Email Message General Information

This is hard to read. The ReadOnly variables are:

- User::CsvFileNameLEA
- User::CsvFileNameSchool
- User::CsvFileNameSEA
- System::PackageName
- User::SuccessEmailCCLine
- User::SuccessEmailToLine
- User::ZipFileNameLEA
- User::ZipFileNameSchool
- User::ZipFileNameSEA

The ReadWrite variables are:

- User::EmailAttachments
- User::EmailMessageBody
- User::EmailMessageCCLine

- User::EmailMessageSubject
- User::EmailMessageToLine

The script that does this task is:

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

    strPackageName = (string)Dts.Variables["System::PackageName"].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;

    if (strZipFileNameSchool == "temp")
    {
        strZipFileNameSchool = "";
        strFileNameSchool = "File not submitted at the School level";
    }

    if (strZipFileNameLEA == "temp")
    {
        strZipFileNameLEA = "";
        strFileNameLEA = "File not submitted at the LEA level";
    }

    if (strZipFileNameSEA == "temp")
    {
        strZipFileNameSEA = "";
        strFileNameSEA = "File not submitted at the SEA level";
    }

    Dts.Variables["User::EmailMessageSubject"].Value =
        strPackageName
        + " processing succeeded";

    strMsgText = "The processing of package '" + strPackageName;
    strMsgText += "' succeeded. \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 ";
    strMsgText += "attached to this email.";
    strMsgText += "\n\nTHEY HAVE NOT YET BEEN SUBMITTED TO USED";

    Dts.Variables["User::EmailMessageBody"].Value = (object)strMsgText;
    Dts.Variables["User::EmailMessageToLine"].Value =
        Dts.Variables["User::SuccessEmailToLine"].Value;
    Dts.Variables["User::EmailMessageCCLine"].Value =
        Dts.Variables["User::SuccessEmailCCLine"].Value;

    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.

24. 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.

25. 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:

General

Parameter Mapping

Result Set

Expressions

Variable Name	Direction	Data Type	Parameter Name	Parameter Size
User::EmailMessageSubject	Input	NVARCHAR	0	-1
User::EmailMessageToLine	Input	NVARCHAR	1	-1
User::EmailMessageCCLine	Input	NVARCHAR	2	-1
User::EmailMessageBody	Input	NVARCHAR	3	-1
User::EmailAttachments	Input	NVARCHAR	4	-1

Screen 54: Submission Loading, Update Email Log, Parameter Mapping

No edits are required from the template.

26. Send Notification

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

Mail	
SmtConnection	MailServer
From	EDFacts@edu.state.gov
To	EDFacts_God@edu.state.gov
Cc	
Bcc	
Subject	Process Failed
MessageSourceType	Variable
MessageSource	User::EmailMessageBody
Priority	Normal
Attachments	

Screen 55: Submission Loading, Send Notification, Mail Settings

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

Expressions	
CCLine	@[User::EmailMessageCCLine]
FileAttachments	@[User::EmailAttachments]
FromLine	@[User::EmailMessageFromLine]
Subject	@[User::EmailMessageSubject]
ToLine	@[User::EmailMessageToLine]

Screen 56: Submission Loading, Send Notification, Expressions

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.

Generic Package Configurations

SSIS configuration information is kept in the database table EDFacts_Admin.SSIS_Configuration. For each package, there are two configuration filters that determine which settings to use for the package execution.

The EDFacts_Admin.SSIS_Configuration table has the following structure:

#	Field Name	Data Type
1	ConfigurationFilter	nvarchar(255)
2	ConfiguredValue	nvarchar(255)
3	PackagePath	nvarchar(255)
4	ConfiguredValueType	nvarchar(20)

The ConfigurationFilter is an arbitrary value that groups settings together for a package configuration. In ES3, the value is either “Core Corrections” for values used by ALL packages, or is set to the package name for the settings specific to a package and that should be exposed at runtime.

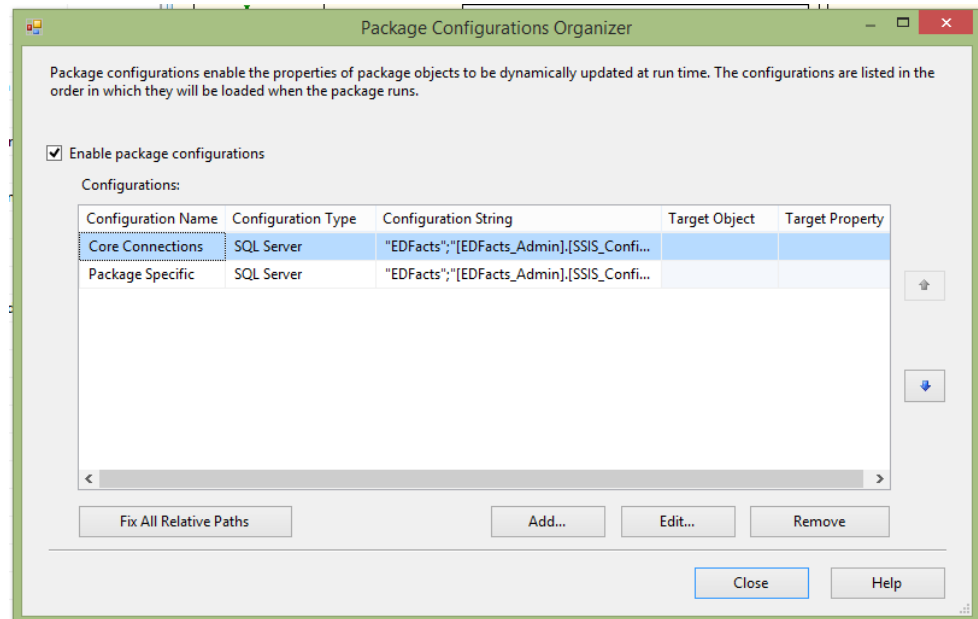
PackagePath is the full name of the value to be set. ConfiguredValue is the value the PackagePath should use. ConfiguredValueType tells the system how to interpret the data type for the ConfiguredValue.

For example, if the package “S052 Membership” exposes the value of the variable “ReportingPeriod” then there is a record in the EDFacts_Admin.SSIS_Configuration table with the following settings:

#	Field Name	Record Entry
1	ConfigurationFilter	S052 Membership
2	ConfiguredValue	2013-2014
3	PackagePath	\Package.Variables[User::ReportingPeriod].Properties[Value]
4	ConfiguredValueType	String

To set the package configuration settings, select “Package Configurations...” from the SSIS menu. Make sure that no task in the SSIS Package control flow tab is currently selected, or the “Package Configuration ...” menu option will not be visible.

The following Package Configuration organizer window will open.



For each package there are two entries: Core Connections and Package Specific.

“Core Connections” Package Configuration

“Package Specific” Package Configuration

SSIS Package Deployment

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

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

Field Name	Data Type	Description
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
hostMachine	varchar(50)	the machine upon which the package is executing – note: this is not the DB against which the package is running necessarily.

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.

General Parameter Mapping Result Set Expressions	<div> <div>General</div> <div>Name</div> <div>Description</div> <div>Options</div> <div>TimeOut</div> <div>CodePage</div> <div>Result Set</div> <div>ResultSet</div> <div>SQL Statement</div> <div>ConnectionType</div> <div>Connection</div> <div>SQLSourceType</div> <div>SQLStatement</div> <div>IsQueryStoredProcedure</div> <div>BypassPrepare</div> </div> <div> <div>OnPreExecute Logging</div> <div>Execute SQL Task</div> <div>0</div> <div>1252</div> <div>None</div> <div>insert into EDFacts_Admin.SSIS_ProcessLog (</div> <div>OLE DB</div> <div>EDFacts</div> <div>Direct input</div> <div>False</div> <div>True</div> </div>
---	---

Screen 58: Logging and Event Handling, OnPreExecute Event, General Information

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

The SQL statement is built in an Expression.

General Parameter Mapping Result Set Expressions	<div> <div>Misc</div> <div>Expressions</div> <div>SqlStatementSource</div> </div> <div> <div>insert into EDFacts_Admin.SSIS_ProcessLog (</div> </div>
---	---

Screen 59: Logging and Event Handling, OnPreExecute Event, Expressions

The expression is:

```
“insert into EDFacts_Admin.SSIS_ProcessLog (
    executionInstanceGUID,
    eventType,
    packageName,
    taskName,
    taskID,
```

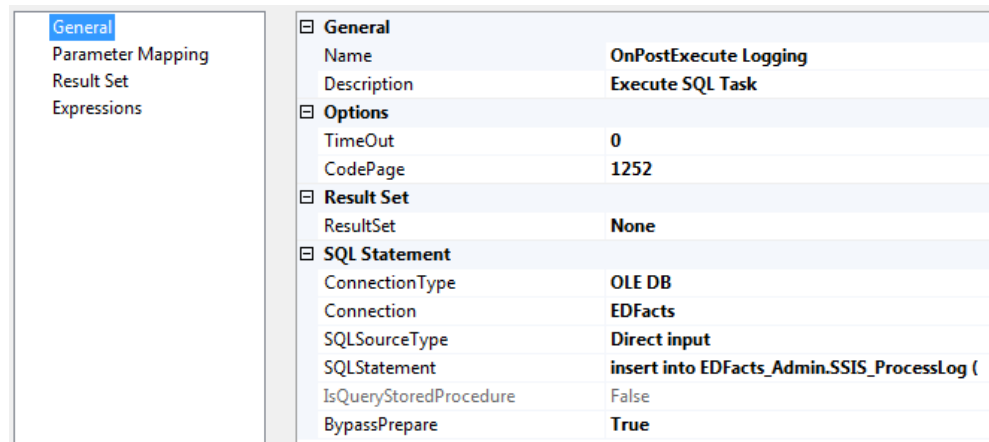
```

        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
    'Starting...', --Task_Status
    '' + @[System::MachineName] + '' --Host_Machine
)

```

OnPostExecute Event Handler

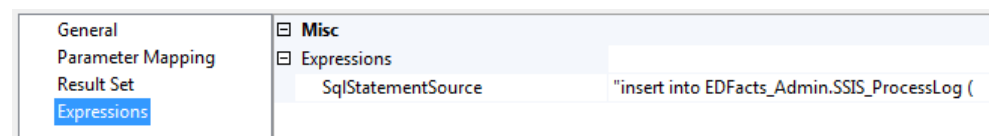
The OnPostExecute event handler is a SQL task that populates another row in the SSIS_ProcessLog table at the end of every task in the package.



Screen 60: Logging and Event Handling, OnPostExecute Event, General Information

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

The SQL statement is built in an Expression.



Screen 61: Logging and Event Handling, OnPostExecute Event, Expressions

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.

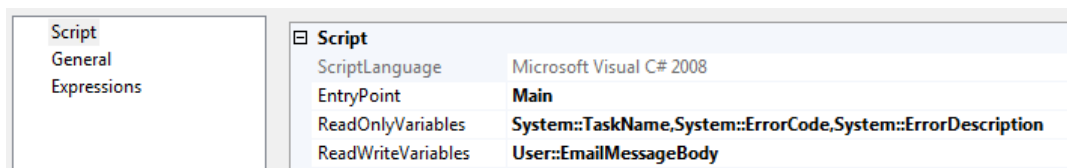
The OnError event handler consists of two tasks:



Screen 62: Logging and Event Handling, OnError Event, Data Flows

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

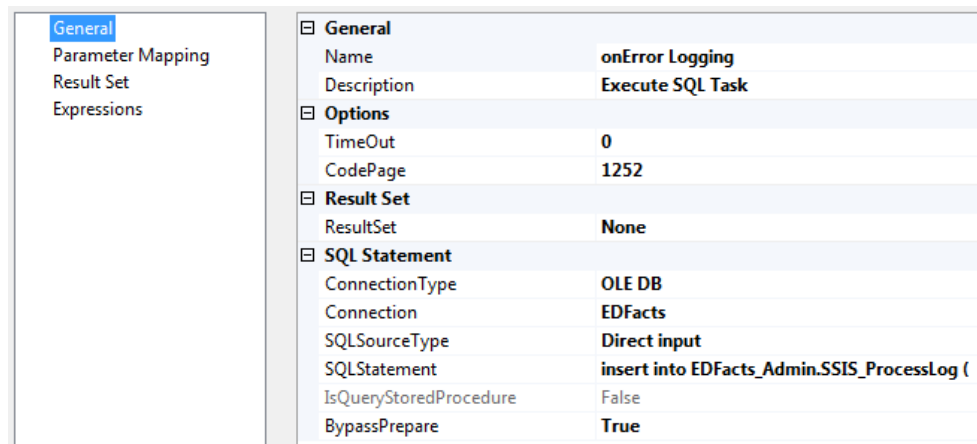


Screen 63: Logging and Event Handling, OnError Event, Update Email Script Information

The script appends this information to the ErrorMessageBody 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::ErrorMessageBody"].Value =
        Dts.Variables["User::ErrorMessageBody"].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;
}
```

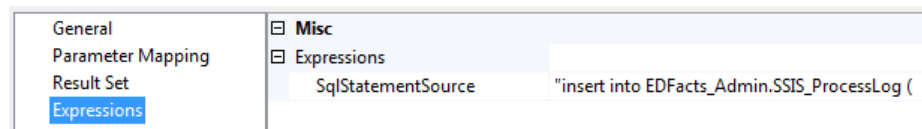
The Error Logging task is a SQL task that inserts a record into the SSIS_ProcessLog table.



Screen 64: Logging and Event Handling, OnError Event, General Information

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

The SQL statement is built in an Expression.



Screen 65: Logging and Event Handling, OnError Event, Expressions

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 to 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. 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_Validation.ValidationResults table that stores the results of the validation checks.

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

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_Validation].[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_Validation.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 (sex, '') = ''
        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 (sex, '') = ''
                    and totalIndicator = 'Y')

end

```

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_Validation].[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 Management System

Individual Client Configuration

Client Configuration Data Tables

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 is 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. The main field of interest is the storage directory root path that identifies the directory under which all the submission files will be stored for a particular reporting period.

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

Field Name	Data Type	Description
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.
storageDirectoryRootPath	varchar(200)	The fully qualified path to the root working directory for files for this reporting period. 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
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 has Charter Schools 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 specification

Field Name	Data Type	Description
dataRecordTableName	varchar(20)	The table name to be used in the data records for this file
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 ES3 system refers to the submission files as Sxxx.

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'
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
description	varchar(1000)	a description, definition, or other note for the particular option.

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)	

Managing State Code Set Translation Values

```

declare @reportingPeriod varchar(9) = ?

delete from EDFacts_Admin.StateCodeTranslation
where codeSetName in ('Age (All)',
                     'Age/Grade (All)',
                     'Age/Grade (w/o under 3)',
                     'Age/Grade (w/o Out of School)',
                     'Age Grade (3-5/K-12)',
                     'Age Group',
                     'Grade Level (Assessment)',
                     'Grade Level (Assessment - Science)',
                     'Grade Level (Basic)',
                     'Grade Level (Dropouts)',
                     'Grade Level (Membership)')
and reportingPeriod = @reportingPeriod

insert into EDFacts_Admin.StateCodeTranslation
(reportingPeriod, codeSetName, stateCode, EDFactsCode, description)
values
-- region      Age (All)
(@reportingPeriod, 'Age (All)', '3T05', '3T05', ''),
(@reportingPeriod, 'Age (All)', '6', '6', ''),
(@reportingPeriod, 'Age (All)', '7', '7', ''),
(@reportingPeriod, 'Age (All)', '8', '8', ''),
(@reportingPeriod, 'Age (All)', '9', '9', ''),
(@reportingPeriod, 'Age (All)', '10', '10', ''),
(@reportingPeriod, 'Age (All)', '11', '11', ''),
(@reportingPeriod, 'Age (All)', '12', '12', ''),
(@reportingPeriod, 'Age (All)', '13', '13', ''),
(@reportingPeriod, 'Age (All)', '14', '14', ''),
(@reportingPeriod, 'Age (All)', '15', '15', ''),
(@reportingPeriod, 'Age (All)', '16', '16', ''),
(@reportingPeriod, 'Age (All)', '17', '17', ''),
(@reportingPeriod, 'Age (All)', '18', '18', ''),
(@reportingPeriod, 'Age (All)', '19', '19', ''),
(@reportingPeriod, 'Age (All)', '20', '20', ''),
(@reportingPeriod, 'Age (All)', '21', '21', ''),
-- end region

-- region      Age/Grade (All)
(@reportingPeriod, 'Age/Grade (All)', 'UNDER3', 'UNDER3', ''),
(@reportingPeriod, 'Age/Grade (All)', '3T05NOTK', '3T05NOTK', ''),
(@reportingPeriod, 'Age/Grade (All)', 'KG', 'KG', ''),
(@reportingPeriod, 'Age/Grade (All)', 'KM', 'KG', ''),
(@reportingPeriod, 'Age/Grade (All)', 'KA', 'KG', ''),
(@reportingPeriod, 'Age/Grade (All)', '01', '01', ''),
(@reportingPeriod, 'Age/Grade (All)', '02', '02', ''),
(@reportingPeriod, 'Age/Grade (All)', '03', '03', ''),
(@reportingPeriod, 'Age/Grade (All)', '04', '04', ''),
(@reportingPeriod, 'Age/Grade (All)', '05', '05', ''),
(@reportingPeriod, 'Age/Grade (All)', '06', '06', ''),
(@reportingPeriod, 'Age/Grade (All)', '07', '07', ''),
(@reportingPeriod, 'Age/Grade (All)', '08', '08', ''),
(@reportingPeriod, 'Age/Grade (All)', '09', '09', ''),
(@reportingPeriod, 'Age/Grade (All)', '1', '01', ''),
(@reportingPeriod, 'Age/Grade (All)', '2', '02', ''),
(@reportingPeriod, 'Age/Grade (All)', '3', '03', ''),
(@reportingPeriod, 'Age/Grade (All)', '4', '04', ''),
(@reportingPeriod, 'Age/Grade (All)', '5', '05', ''),
(@reportingPeriod, 'Age/Grade (All)', '6', '06', ''),
(@reportingPeriod, 'Age/Grade (All)', '7', '07', ''),
(@reportingPeriod, 'Age/Grade (All)', '8', '08', ''),
(@reportingPeriod, 'Age/Grade (All)', '9', '09', ''),
(@reportingPeriod, 'Age/Grade (All)', '10', '10', ''),
(@reportingPeriod, 'Age/Grade (All)', '11', '11', ''),
(@reportingPeriod, 'Age/Grade (All)', '12', '12', ''),
(@reportingPeriod, 'Age/Grade (All)', '00S', '00S', ''),
(@reportingPeriod, 'Age/Grade (All)', 'UG', 'UG', ''),
(@reportingPeriod, 'Age/Grade (All)', '?', '<ignore>', ''),

```



```
(@reportingPeriod, 'Age/Grade (All)', 'UNKNOWN', 'UG', ''),  
--end region
```

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 >

Standards and Best Practices

Naming conventions

T-SQL procedure naming and comment conventions