Migration Utility

Migration Utility v2.0.0

September 2020: On behalf of the Ed-Fi community, we are pleased to announce the release of ODS Migration utility 2.0.0 which includes support for upgrade to the latest Ed-Fi ODS / API v5.0.0 version.

Downloads

Ed-Fi ODS Migration Utility binaries: https://www.myget.org/feed/ed-fi/package/nuget/EdFi.Suite3.Ods.Utilities.Migration/2.0.0
(Prerequisite: .NET Core 3.1)

Ed-Fi ODS Migration Utility source code, hosted on Ed-Fi Alliance GitHub: https://github.com/Ed-Fi-Alliance-OSS/Ed-Fi-MigrationUtility

Example calendar configuration files: Sample Calendar Config

Overview

The ODS Migration Utility is a command-line tool built to upgrade the schema of an ODS instance up to the latest version, along with data migration.

It currently supports data migration from Ed-Fi Data Standard v2.0 and Ed-Fi Data Standard v2.2 to Ed-Fi Data Standard v3.2. The utility has out-of-the-box support for migrating an as-shipped ODS to the latest version. With additional customized scripting, the Migration Utility can be easily adapted and used to migrate extended ODS instances. ODS shared instances may take advantage of this utility. For year-specific instances, migration may not be a concern as a new ODS is created at the beginning of every school year.

Contents:

Usage Scenarios

The following table summarizes the supported scenarios for the migration utility:

Database Type	Databases	Upgrade/Migration strategy
Core Databases Databases that surface the Ed-Fi model and store user data.	EdFi_Ods, EdFi_Ods_YYYY, EdFi_Ods_Sandbox	The current migration utility release supports an in-place upgrade for the following upgrade paths: 2.4 -> 5.0.0 2.5 -> 5.0.0 3.0 -> 5.0.0 The migration utility also supports migrating extensions with additional custom scripts.
Support Databases Databases that provide supporting functions and store user data.	EdFi_Admin, EdFi_Security, EdFi_Bulk	These databases can either be recreated, or existing Entity Framework Migrations can be used to migrate them.
Transient Databases Databases that either surface the Ed-Fi model or perform supporting functions, but do not have user data persistence requirements.	EdFi_ODS_Empty, EdFi_Ods_Minimal_Template EdFi_Ods_Populated_Template	No upgrade supported.

Developer Quick Start

The basic steps are simple:

- Restore a backup copy of the target ODS to your local SQL Server instance. We recommend:
  - Start with the basic suite 2 ODS: Sample ODS Download: EdFi.Samples.Ods/2.0.0.21. This is a small sample ODS, ideal for initial development and testing.
  - Move on to the larger suite 2 ODS: Glendale 2.0 ODS Backup (created 2018-06-13). This is a much larger ODS containing sample data, useful for validation and QA.
- Make sure .NET Core 3.1 is installed.
- Choose one of the two options below to launch the migration utility:

Step 1. Install the Ed-Fi ODS Migration tool:

```
c:\>mkdir {YourInstallFolder}
c:\>dotnet tool install EdFi.Suite3.Ods.Utilities.Migration --tool-path
{YourInstallFolder}
```

Note: As onetime setup, you may need to add Ed-Fi MyGet package source by running the following command in Powershell:

```
Register-PackageSource -Name Ed-FiMyget -Location https://www.myget.org
/F/ed-fi/ -ProviderName NuGet
```

Step 2. Open a console window and change to the directory containing the executable

```
CD {YourInstallFolder}
```

Step 3. Launch the upgrade tool from the command line

Example Command Line Arguments: Grand Bend ODS

```
.\EdFi.Ods.Utilities.Migration --DATABASE
"YOUR_DATABASE_CONNECTION_STRING_HERE" --DescriptorNamespace
"uri://grandbend.org" --CredentialNamespace "uri://grandbend.org"
```

Step 1. Clone git repository: https://github.com/Ed-Fi-Alliance/Ed-Fi-MigrationUtility

Step 2. Open the Visual Studio solution file, `Migration.sln`.

Step 3. Set up command line input for debugging in Visual Studio 2019:

- Right click the `EdFi.Ods.Utilities.Migration` project.
- Select **properties**.
- Select **debug**.
- Add command line arguments:

Example Command Line Arguments: Grand Bend ODS

```
--DATABASE "YOUR_DATABASE_CONNECTION_STRING_HERE" --DescriptorNamespace
"uri://grandbend.org" --CredentialNamespace "uri://grandbend.org"
```

Step 4. Set the `EdFi.Ods.Utilities.Migration` project as your startup project.

Step 5. Launch in debug mode (**F5**).

Development Overview: The Basics

The table below describes files and folders used by the Migration Utility along with a description and purpose for each resource.

Overview Item	Needed by Whom?	Brief Description & Purpose

Script Directory: \Scripts	• **Users writing custom upgrade scripts**
	• Maintainers of the upgrade utility
		• All database upgrade scripts go here, including:
		  ◦ Ed-Fi upgrade scripts.
		  ◦ Custom upgrade scripts (user extensions).
		  ◦ Compatibility checks.
		  ◦ Dynamic / SQL-based validation.
		• Subdirectories contain code for each supported ODS upgrade version.

Directory: \Descriptors	• Maintainers of the upgrade utility only	• Ed-Fi-Standard XML files containing Descriptors for each ODS version. • These XML files are imported directly by the scripting in the `Utilities\Migration\Scripts` directory above.

Library/Console: EdFi.Ods.Utilities.Migration (console application created via dotnet publish)	• Maintainers of the upgrade utility only	• Small, reusable library making use of DbUp to drive the main upgrade. • Executes the SQL scripts contained in the `Utilities\Migration\Scripts` directory above. • Takes a configuration object as input, and chooses the appropriate scripts to execute based on ODS version and current conventions.
Test Project: EdFi.Ods.Utilities.Migration.Tests	• Maintainers of the upgrade utility only (does not contain test coverage for extensions)	• Contains integration tests that perform a few test upgrades and assert that the output is as expected. • Like the console utility, makes direct use of the `EdFi.Ods.Utilities.Migration` library described above.

Development Troubleshooting

This section outlines general troubleshooting procedures.

Compatibility Errors

- Before the schema is updated, the ODS data is checked for compatibility. If changes are required, the upgrade will stop and exception will be thrown with instructions on how to proceed.
  An example error message follows:

```
--------------------------------------------------------------------------------------------------------------------------------
Action Required - [edfi].[Assessment]: All assessments must have a [Namespace] set.  (This data may be found in [edf
Assessment] or [edfi].[AssessmentFamily]).  This data is required by schema beginning in version 3.0. (Message Class
010)
--------------------------------------------------------------------------------------------------------------------------------
Press any key to continue . . .
```

- After making the required changes (or writing custom scripts), simply launch the upgrade utility again. The upgrade will proceed where it left off and retry with the last script that failed.

Other Exceptions During Development

- Similar to compatibility error events, the upgrade will halt and an exception message will be generated during development if a problem is encountered.
- After making updates to the script that failed, simply re-launch the update tool. The upgrade will proceed where it left off starting with the last script that failed.
- If you are testing a version that is not yet released, or if you need to re-execute scripts that were renamed or modified during active development: restore your database from backup and start over.
- Similar to other database migration tools, a log of scripts successfully executed will be stored in the default journal table. DbUp's default location is `[dbo].[SchemaVersions]`.
- A log file containing errors/warnings from the most recent run may be found by default in `C:\ProgramData\Ed-Fi-ODS-Migration\Migration.log`.

Additional Troubleshooting

- The step-by-step usage guide below contains runtime troubleshooting information.

Design/Convention Overview

The table below outlines some important conventions in the as-shipped Migration Utility code.

What	Why	Optional Notes
In-place upgrade Database upgrades are performed in place rather than creating a new database copy	**Extensions** • Preserve all unknown data in extension tables • Ensure errors and exceptions are properly generated and brought to the upgrader's attention during migration if the upgrade conflicts with an extension or any other customization As a secondary concern, this upgrade method was chosen to ease the upgrade process for a Cloud-based ODS (e.g., on Azure).	• Users who have extension tables (or any other schema with foreign key dependencies on the `edfi` schema) will be notified, and must explicitly acknowledge it by adding the `BypassExtensionValidationCheck` option at the command line when upgrading. This will ensure that the installer is aware that custom upgrade scripts may be required

Sequence of events that occur during upgrade
 	
Specifics differ for each version, but in general the upgrade sequence executes as follows
 	
1. Validate user input at the command line
2. Create tempdata (tables/stored procedures) that will be used for upgrade
3. Check current ODS data for upgrade compatibility, and display action messages to the user if existing data requires changes
4. Before modifying the `edfi` schema: calculate and store hash values for primary key data that expected **NOT** to change during upgrade
5. Drop views, constraints, stored procs
6. Import descriptor data from XML
7. Create all new tables for this version that did not previously exist
8. Update data in existing tables
9. Drop old tables
10. Create views /constraints/stored procs for the new version
11. Validation check: recalculate the hash codes generated previously, and make sure that all data that is not supposed to change was not mistakenly modified
12. Drop all temporary migration data
	Minimize the number of scripts with complex dependencies on other scripts in the same directory /upgrade step.
	
	• Compatibility checks are designed to run before any changes to the `edfi` schema have been made. This prevents the user from having to deal with a half-upgraded database while making updates
	• Initial hash codes used for data validation also must be generated before touching the `edfi` schema to ensure accuracy.
	  • It is also better for performance to do this step while all of our indexes are still present
	• Dropping of constraints, views, etc is taken care of before making any schema changes to prevent unexpected sql exceptions
	• New descriptors are imported as an initial step before making changes to the core tables. This ensures that all new descriptor data is available in advance for reference during updates
	• After creation of descriptors, the sequence of the next steps is designed to ensure that all data sources exist *unmodified on the old schema* where we expect it to.
	  1. Create tables that are **brand new** to the schema only (and populate them with existing data)
	  2. Modify existing tables (add/drop columns, etc)
	  3. Drop old tables no longer needed
	• Foreign keys, constraints, etc are all added back in once the new table structure is fully in place.
	• Once the `edfi` schema is fully upgraded and will receive no further changes, we can perform the final data validation check.
		The suite 2 to suite 3 upgrade is a good example case to demonstrate the upgrade steps working together due its larger scale:
		
		• For suite 2 to suite 3: All foreign keys and other constraints were dropped during this upgrade in order to adopt the new naming conventions
		• Also for suite 2 to suite 3: ODS types were replaced with new descriptors. This change impacted nearly every table on the existing schema

One script per table in each directory, where possible Scripts are named in the format: `#### TableName [optional_tags].sql` This convention does not apply to operations that are performed dynamically	**Troubleshooting, Timeout prevention** Custom, unknown extensions on the ODS are common. As part of the process of upgrading a highly-customized ODS, an installer is likely to run into a sql exception somewhere in the middle of upgrade (usually caused by a foreign key dependency, schema bound view, etc). In general, we do not want to attempt to modify an unknown/custom extension on an installer's behalf to try and prevent this from happening. It is important that a installer be aware of each and every change applied to their custom tables. Migration of custom extensions will be handled by the installer. Considering the above, in the event an exception does occur during upgrade, we want to make the troubleshooting process as easy as possible. If an exception is thrown, an installer should immediately be able to tell: • Which table was being upgraded when it occurred (from the file name) • What were the major changes being applied (from the file tags) • What went wrong (from the exception message) • Where to find the code that caused it Many issues may be fixable from the above information alone. If more detail is needed, the installer can view the code in the referenced script file. By separating script changes by table, we make an effort to ensure that there are only a few lines to look though (rather than hundreds) In addition, each script will be executed in a separate transaction. Operations such as index creation can take a long time on some tables with a large ODS. Splitting the code into separate transactions helps prevent unexpected timeout events	The major downside of this approach is the large number of files it can produce. For example, the suite 2 to suite 3 upgrade was a case where all existing tables saw modifications. This convention generates a change script for every table in more than one directory. With updates becoming more frequent in the future, future versions should not be impacted as heavily.
Most change logic is held in sql scripts (as of V3) As of v3: Most of the upgrade logic is performed from the SQL scripts, rather than using .NET based upgrade utility to write database changes directly	As of v3, most upgrade tasks are simple enough where they can be executed straight from SQL (given a few stored procedures to get started). Given this advantage, effort was made to ensure that each part of the migration tool (console utility, library, integration tests) could be replaced individually as needed The current upgrade utility contains a library making use of DbUp to drive the upgrade process. In the future, if/when this tool no longer suits our needs, we should be able to take existing scripting and port it over to an alternative upgrade tool (such as RoundhousE), or even a custom built tool if the need ever arises. This convention could (and should) change in the future if upgrade requirements become too complex to execute from SQL scripting alone.	

Two types of data validation/testing options
 	
• Dynamic, SQL based
  • Ensures data that is expected to remain the same does not change
  • Can run on any ODS in the field even if the data is unknown
• Integration tests
  • Runs on a known, given set of inputs
  • Used to test logic in areas where changes should occur
	Prevent data loss
	
	The first type of validation, (dynamic, sql based) is executed on on data that we know should **not** ever change during the upgrade.
	
	• The source and destination tables do not need to be the same. This validation type is most commonly used to verify that data was correctly moved to the expected destination table during upgrade
	• Can be executed on any field ODS to ensure that unknown datasets do not cause unexpected data loss during upgrade
	• The data in these tables does not need to be known
	
	The second type of data validation, integration test based, is used to test the logic and transformations where we know the data **should** change:
	
	• For example, during the suite 2 to suite 3 upgrade, descriptor namespaces are converted from the suite 2 "`http://EdOrg/Descriptor/Name.xml`" format to the suite 3 "`uri://EdOrg/Name`" format. Integration tests are created to ensure that the upgrade logic is functioning correctly for several known inputs
	
	Together, the two validation types (validation of data that changes, and validation of data that does not change) can be used to create test coverage wherever it is needed for a given ODS upgrade.
		The the dynamic validation is performed via reusable stored procedures that are already created and available during upgrade.
		
		See scripts in the "`*Source Validation Check`" and "`*Destination Validation Check`" directories for example usages.

Upgrade Issue Resolution Approach

It is common to encounter scenarios where data cannot be upgraded directly from one major version to another due to schema changes. A common example is that a primary key change causes records allowable by the previous version schema to be considered duplicates on the upgrade version schema and therefore not allowed.

The general approaches included here are a result of collaboration with the community on how to resolve these common situations, and are documented here as a reference for utility developers to apply in their own work.

- **Approach 1**: Make non-breaking, safe changes to data on the user's behalf. *This is the preferred option when practical and safe to do so.*
  - Example: Duplicate session names in the suite 2 schema that will break the new suite 3 primary key will get a term name appended during upgrade to meet schema requirements.
  - Consider logging a warning for the user to review to inform them that a minor change has taken place, and mention the log file in documentation.
- **Approach 2**: Throw a compatibility exception asking the user to intervene with updates. *Used when we are unable to safely make assumptions on the user's behalf to perform the update for them*
  - See the troubleshooting section of this document for an example of this
  - The returned user-facing message should explain what updates are required, and very briefly include the reason why if it is not immediately obvious.
  - Compatibility exceptions should be thrown before proceeding with core upgrade tasks.
    - We want to make sure that we are not asking someone to make updates to a database in an intermediate upgrade state.
  - Each message class has a corresponding error code, mirrored in code by an enumeration
    - This is done so that specific compatibility scenarios can be covered by integration tests where needed
- **Approach 3**: Back up deprecated objects that cannot be upgraded into another schema before dropping them. *Last resort option that should be mainly restricted to deprecated items that are no longer have a place in the new model*
  - As a general approach, it is preferred to avoid dropping data items without a way to recover them in case of disaster. The user may choose to delete the backup if they desire.
  - Avoid this option for tables that exist in both models but simply cannot be upgraded. Should this (*hopefully rare*) situation occur, consider the option throwing a compatibility exception instead and ask the user back up/empty the table before proceeding.

In general, the option requiring the least amount of user effort while safely preserving all data has been chosen in order to reduce user burden as much as we are able.

Usage Walkthrough

This section explains how to upgrade an existing suite 2 ODS to suite 3 ODS v5.0.0.

The steps can be summarized as:

A compatibility reference chart, a command-line parameter list, and a troubleshooting guide are included. Each upgrade step is outlined in detail below.

Step 1. Read the Ed-Fi ODS v5.0.0 Upgrade Overview

Before you get started, you should review and understand the information in this section.

Target Audience

These instructions have been created for technical professionals, including software developers and database administrators. The reader should be comfortable with performing the following types of tasks:

- Creating and managing SQL Server database backups
- Performing direct SQL commands and queries
- Execution of a command-line based tool that will perform direct database modifications
- Creating a configuration file for upgrade (.csv format)
- Writing custom database migration scripts (Extended ODS only)

General Notes

- **Your suite 2 Ed-Fi ODS / API will checked for compatibility automatically during the migration process.** If changes are needed, you will be prompted at the command line by the migration utility. A summary of commonly encountered compatibility conditions has been included in this section for reference.
- The new schema may contains upgrades to the structure of primary keys on several tables. In most instances, these new uniqueness requirements will be resolved automatically for you with no action required.
- There are some areas where new identities cannot be generated automatically on your behalf during upgrade. These tables will need to be updated manually.

Compatibility Conditions

This section describes compatibility conditions (i.e., requirements that may need intervention for the compatibility tool to function properly) and suggested remediation.

Table	Data Compatibility Requirement
`[edfi].[Assessment]`	All assessments must have a `[Namespace]` set. (This data may be found in `[edfi].[Assessment]` or `[edfi].[AssessmentFamily]`).

[edfi].[StudentProgramParticipation] [edfi].[StudentCharacteristic] [edfi].[StudentIndicator] [edfi].[StudentLearningStyle] [edfi].[StudentAddress] [edfi].[StudentIdentificationCode] [edfi].[StudentElectronicMail] [edfi].[StudentInternationalAddress] [edfi].[StudentLanguage] [edfi].[StudentRace] [edfi].[StudentDisability] [edfi].[StudentTelephone] [edfi].[PostSecondaryEventPostSecondaryInstitution]	The upgrade utility must be able to locate an **[EducationOrganizationId]** for every student with data in the listed tables to proceed. *The easiest way to meet this requirement is to ensure that every student has a corresponding record in [edfi].[StudentSchoolAssociation] or [edfi].[StudentEducationOrganizationAssociation].*
[edfi].[StaffCredential]	The column **[StateOfIssueStateAbbreviationTypeId]** must be non-null for all records. This is the abbreviation for the name of the state (within the United States) or extra-state jurisdiction in which a license/credential was issued.
(Any extension table)	Additional steps are required when extensions are present. Please review the upgrade process detailed below for additional guidance.

Table	Data Compatibility Requirement
[edfi].[GradingPeriod]	• There must be no duplicate **[PeriodSequence]** values for the same school during the same grading period. • If prompted by the upgrade utility, all **[PeriodSequence]** values must be non-null *Technical Details:* *This compatibility requirement is a result of a primary key change between suite 2 and suite 3 version 3.1* • *Old 2.0 Primary Key: [GradingPeriodDescriptorId], [SchoolId], [BeginDate]* • *New 3.1 Primary Key: [GradingPeriodDescriptorId], [SchoolId], **[PeriodSequence]** (new), **[SchoolYear]** (new). ([BeginDate] is removed)*
[edfi].[DisciplineActionDisciplineIncident]	The 3.1 schema no longer allows discipline action records with students that are not associated with the discipline incident. Every record in [edfi].[DisciplineActionDisciplineIncident] must have a corresponding record in [edfi].[StudentDisciplineIncidentAssociation] with the same [StudentUSI], [SchoolId], and [IncidentIdentifier].
[edfi].[RestraintEvent]	Ensure that there are no duplicate **[RestraintEventIdentifier]** values for the same student at the same school
[edfi].[OpenStaffPosition]	Ensure that there are no two duplicate **[RequisitionNumber]** entries for the same education organization
[edfi].[AccountCode]	This table must be empty before upgrading. Due to a major schema change, data in this table cannot be preserved during upgrade from suite 2 to suite 3

Other Compatibility Conditions

There are several other less common items not included above. The migration utility will check for these items automatically and provide guidance messages as needed. For additional technical details, please consult the Troubleshooting Guide below.

Step 2. Install Required Tools

- Ensure .NET Core 3.1 is installed.
- Add Ed-Fi MyGet package source by running the following command in Powershell:

```
Register-PackageSource -Name Ed-FiMyget -Location https://www.myget.org/F
/ed-fi/ -ProviderName NuGet
```

Step 3. Uninstall the previous version Ed-Fi ODS Migration Utility if any

- The package id of the binaries changed to include suite number, If you had previously installed 'EdFi.Ods.Utilities.Migration' package, uninstall it from a PowerShell prompt using the following command:

Install Ed-Fi ODS Migration Utility

```
c:\>cd {YourAlreadyInstalledFolder}
c:\>dotnet tool uninstall EdFi.Ods.Utilities.Migration --tool-path
{YourAlreadyInstalledFolder}
You will receive below message for uninstalled is successful
"Tool 'edfi.ods.utilities.migration' (version '1.1.0') was successfully
uninstalled."
```

Step 4. Install the Ed-Fi ODS Migration Utility

- Install the Ed-Fi ODS Migration utility from a PowerShell prompt using the following command:

Install Ed-Fi ODS Migration Utility

```
c:\>mkdir {YourInstallFolder}
c:\> dotnet tool install EdFi.Suite3.Ods.Utilities.Migration  --version
2.0.0 --tool-path {YourInstallFolder}
You will receive below message for installed is successful
Tool 'edfi.suite3.ods.utilities.migration' (version '2.0.0') was
successfully installed.
```

Step 4. Back Up and Create a Working Copy of the Suite 2 Target Database

- Create a **full backup** of the target **suite 2 EdFi_Ods** database.
- Restore this backup to your SQL Server instance as a copy, in a new location.
- Make note of the database connection string as it will be required for a later step.

Step 5. (Multi-Year ODS Only) Create a Calendar Configuration File

In most cases, you can skip this step if the ODS contains valid calendar data for only one school year. The migration tool will usually be able to resolve this item automatically for you.

The suite 3 ODS upgrade includes enhancements to the calendar, including tracking of school year information for every calendar event and session. To ensure that your new calendar is accurate, the migration tool will need to know which school year to associate with every calendar item in your ODS.

Example calendar configuration files can be downloaded from the **download** panel on the right.

The following is an example of a calendar configuration file (CSV) for Grand Bend ISD:

Example 1: Grand-Bend calendar-config.csv

```
SchoolId,SchoolYear,StartDate,EndDate
255901001,2011,2010-08-23,2011-05-27
255901044,2011,2010-08-23,2011-05-27
255901107,2011,2010-08-23,2011-05-27
```

Note that each line contains a SchoolId, SchoolYear, the first calendar day and the last calendar day of the school year.

- Create a new calendar configuration file (.CSV) by referencing the above examples. Your configuration file must match the following format (**including header row**):

Example 2: calendar-config.csv

```
SchoolId,SchoolYear,StartDate,EndDate
{Your_First_School_Id},20XX,{FirstDayOfSchoolYear},{LastDayOfSchoolYear}
{Your_Second_School_Id},20XX,{FirstDayOfSchoolYear},{LastDayOfSchoolYear}
...
```

- Store the configuration file in a location that is accessible by your SQL server

Step 6. (ODS with NO Extensions Only) Run the Migration Utility

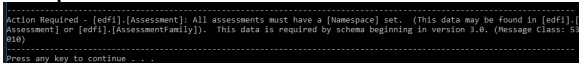
- Open a new command prompt and navigate to the Ed-Fi ODS Migration utility installation folder:

```
CD {YourInstallFolder}
```

- Launch the console tool. Modify the below example parameters to match your environment. Replace the example "`uri://ed-fi.org`" with the Namespace Prefix to insert for your new descriptors:

```
.\EdFi.Ods.Utilities.Migration --Database
"YOUR_DATABASE_CONNECTION_STRING_HERE" --CalendarConfigPath "C:
\PATH\TO\YOUR\CALENDAR_CONFIG_IF_APPLICABLE.csv" --
DescriptorNamespace "uri://ed-fi.org" --CredentialNamespace
"uri://ed-fi.org"
```

- Notes for upgrades performed on a remote database
  - The calendar configuration file path provided during upgrade must be accessible **by your SQL server** for importing and validation.
  - The XML files used to create suite 3 descriptors are by default located in "`{YourInstallFolder\.store\edfi.suite3.ods.utilities.migration\2.0.0\edfi.suite3.ods.utilities.migration\2.0.0\tools\netcoreapp3.1\any\Descriptors\3.1`. **This directory must also be accessible by your SQL server** for importing. If you need to copy this directory to a new location, you may make use of the `--DescriptorXMLDirectoryPath "C:\PATH\TO\YOUR\DESCRIPTOR\XML"` parameter to point the migration tool to the new location of these items
- Users who are running the migration tool on an extended ODS **must** use the `--BypassExtensionValidationCheck` option during upgrade in order to allow the tool to make changes when extension dependencies are present.
  - *Details on upgrading an extended ODS are included in the steps below.*
- The console tool will check your ODS for compatibility, and then proceed to perform an in-place upgrade on the specified database.
- **If you encounter any compatibility messages or errors during upgrade**
  - During the upgrade process, your ODS will be checked for compatibility with the latest version. If changes are required, you may encounter a compatibility message, and the upgrade will stop.
    Example



    - After making the required changes (or writing custom scripts), simply launch the upgrade utility again. The upgrade will proceed where it left off and retry with the last script that failed.
    - See the Troubleshooting Guide below for additional guidance
- **Once the process runs to completion with no errors, your upgrade is complete**
- For a summary of all available parameter options for the Ed-Fi ODS console-based migration tool, please see the included reference below.

Step 7. (Extended/Modified ODS Only) - Perform a Test Migration Without Extensions

- It is highly recommended that you first perform a test migration without your extension tables present in order to ensure that all core upgrade requirements have been met
    - On your ODS copy, temporarily drop all extension tables.
    - Complete the migration process exactly as detailed in Step 6, above.
    - Once the migration has completed successfully on the core Ed-Fi data, restore your suite 2 ODS working copy from backup, including all extension tables, and proceed with the next step

Step 8. (Extended/Modified ODS Only) - Write Custom Migration Scripts for Extensions

Important notes before you begin

- When running the migration tool on an ODS with extensions or external dependencies on the EdFi schema, you **must** add the **`--BypassExtensionValidationCheck`** parameter at the command line. This is required in order to permit the migration tool to make changes to an extended database.
- Migration utility may drop and re-create auth views if changes are needed for the upgraded ODS. If you have modified auth views, review and reapply your changes after migration as necessary.
- The Troubleshooting Guide below contains helpful advice for users upgrading an ODS with extensions.

Step 8a. Locate the Ed-Fi Migration scripts and review the current upgrade conventions

- The edfi migration scripts are located by default in **{YourInstallFolder}\.store\edfi.suite3.ods.utilities.migration\2.0.0\edfi.suite3.ods.utilities.migration\2.0.0\tools\netcoreapp3.1\any\Scripts**
- All SQL scripts are run in numerical order, with version-specific upgrade further categories.
- The v2.0 to V3.1 upgrade passes though v2.5 on the way. It will execute scripts in the following version directories:
    - v24_to_v25
    - v25_to_v31

The directory structure appears as follows:

Directory Name	What does this directory contain?	What scripts should you add here?
00 Setup	The first group of scripts that is run before data migration begins. Setup-type scripts, such as the creation of stored procedures used during upgrade are added here.	**Optional:** You may place custom scripts that you would like to execute **before compatibility checks or schema updates begin** here. Scripts in each version directory will be executed in numerical order.
01 Compatibility Check	Ed-Fi data compatibility preconditions are checked using the scripts in this directory.	**None needed.**
02 Upgrade	Contains the core upgrade scripts for each version. All SQL scripts are run in numerical order by path. Details have been included below this chart. The v2.0 to v3.1 upgrade will execute scripts in the following version directories:	**Optional:** - You may place custom scripts that you would like to execute on your **v2.0** database in directory: **`02 Upgrade\v24_to_v25\{step}`**.

	• v24_to_v25 • v25_to_v31	• Scripts that you want to execute on a **v2.5** database may be placed in directory: **`02 Upgrade\v25_to_v31\{step}`**. • Scripts in each version directory will be executed in numerical order. • Details on version-specific upgrade steps are included below this table.
03 Cleanup	Removes temporary data generated during upgrade	**Optional:** You may place custom scripts that you would like to execute **after all schema updates have been performed** here. Scripts in will be executed in numerical order order.

Version-Specific Upgrade Step Details for suite 2 to Suite 3 v5.0.0

The core upgrade script directory, 02 Upgrade, contains several upgrade steps for each version which are detailed below. You can insert custom scripts at any point in the process as needed for custom Extensions.

	Step /Directory Name	What is contained
1	Source Validation Check	Calculates a hash value for data key columns that are to remain unchanged throughout the migration process. The hash will be recalculated after all schema changes have been completed. Values before and after are compared. In the event of a data mismatch, the migration process will be halted automatically to protect data integrity.
2	Create Upgrade Tempdata	Version-specific temporary data is created here that will be removed at the end of the upgrade process.
3	Drop Extended Properties	Drops all extended properties on the `edfi` schema before updates begin.
4	Drop Views	Contains scripts that have been automatically generated to drop `edfi` views. You may choose to insert scripts here to drop custom schema bound custom views before upgrade tasks begin.
5	Drop Procedures, Types	Contains scripts that have been automatically generated to drop `edfi` defau lt stored procedures, triggers, and types.
6	Drop Constraints	Contains dynamic scripting that drops ALL 2.0 constraints and indexes on the `edfi` schema. Objects are dropped in the following order: 1. Foreign Keys 2. Primary Keys 3. Unique Constraints 4. Default Constraints 5. Indexes **Required for most extensions:** You must drop ALL foreign keys on your extension tables that have a dependency on the `edfi` schema. Number your scripts such that they run before the scripts are executed. This will prevent a sql exception from occurring when dependent items on the `edfi` schema are dropped. You may either perform this action during the setup step, or by inserting custom scripts in this directory. See the below sections for tips and details.
7	Import Descriptors	Imports new Ed-Fi descriptors from the included xml files.
8	Create New Tables	Creates tables that are brand new in v5.0.0 Populates new tables with existing data that has moved in the latest version where applicable.
9	Update Existing Tables	Makes schema changes to tables that exist in both the old and the new versions.

10	Drop Old Tables	Drops tables no longer used in v5.0.0.
11	Create Constraints	Contains scripts that are automatically generated. Creates v5.0.0 indexes, constraints, primary keys, and foreign keys.
12	Create User Roles	Contains script from the v5.0.0 ODS build to insert default users.
13	Create Views	Contains scripts from the v5.0.0 ODS build to create v5.0.0 views.
14	Create Procedures	Contains scripts from the v5.0.0 ODS build to create v5.0.0 stored procedures.
15	Create Extended Properties	Contains scripts that are automatically generated. Adds extended properties to all `edfi` objects
16	Destination Validation Check	Recalculates hash values for all items that were created during step: `Source Validation Check`. In the event that a data change is found, the migration process will halt and an error message will be thrown.

Step 8b. Write scripts to drop ALL foreign keys on your extension tables that depend on the edfi schema

- All edfi primary keys and indexes are dropped automatically during upgrade. The existence of any external constraint that depends on edfi data will result in an error state.
- You may insert these scripts in the following directory
  - Option 1: Insert with similar Ed-Fi scripts in directory: `Scripts/02 Upgrade/{version}/## Drop Constraints`
  - Option 2: (To run before all upgrade tasks begin): `Scripts/00 Setup/{version}`
- Scripts will be executed in numerical order. Number your scripts such that they run before the edfi scripts are executed. This will prevent a sql exception from occurring when dependent items on the `edfi` schema are dropped.
- **Tip:** The following query can quickly identify all constraints that need to be dropped for migration to proceed. Use this as a guide when writing your custom migration scripts.

Quickly show all external dependencies that will need to be dropped

```
SELECT DISTINCT
                parentSchema.name AS 'External Schema Name',
                parentObject.name AS 'Table Name',
                constraintObject.name AS 'Constraint Name',
                CONCAT('References ', referencedSchema.name, '.',
referencedObject.name) AS [Conflict Reason],
                CONCAT('ALTER TABLE ', parentSchema.name, '.',
parentObject.name, ' DROP CONSTRAINT ',  constraintObject.name) AS
[Example Code For Reference]
        FROM sys.foreign_key_columns fk
        INNER JOIN sys.objects parentObject
        ON fk.parent_object_id = parentObject.object_id
        INNER JOIN sys.objects referencedObject
        ON fk.referenced_object_id = referencedObject.object_id
        INNER JOIN sys.schemas parentSchema
        ON parentObject.schema_id = parentSchema.schema_id
        INNER JOIN sys.schemas referencedSchema
        ON referencedObject.schema_id = referencedSchema.schema_id
        INNER JOIN sys.objects constraintObject
                ON constraintObject.object_id = fk.constraint_object_id
        WHERE referencedSchema.name = 'edfi'
                AND parentSchema.name NOT IN ('edfi', 'migration_tempdata')
                AND parentObject.type = 'U'
                AND referencedObject.type = 'U'
```

Step 8c. Create Other Migration Scripts as Needed for Custom Extensions

- After creating scripts that drop all dependent constraints, you can re-test your data migration to ensure that it will complete without error. Be sure to add the `--BypassExtensionValidationCheck` option at the command line.

- Create the remaining scripts needed to upgrade your extension tables to suite 3 using the chart provided above for reference.

Step 8d. Create Changed Record Queries Migration Scripts for Custom Extensions (Required only if the feature was enabled)

- Changed Record Queries is an optional feature that was introduced in ODS 3.1. Changed Record Query artifacts for the core entities will be upgraded automatically by the migration utility if the feature was enabled in the source database. However if you are using Changed Record Query feature in the source ODS and you have custom extensions, then the change scripts to update Changed Record Query artifacts for the extension tables will need to be added into the scripts folder located in Scripts/02 Upgrade/{Version}_Changes.

Step 8e. Upgrade Your Extended ODS

- With all custom scripting in place, you may now proceed with the upgrade as detailed in Step 6. (ODS with NO Extensions Only) Run the Migration Tool, above.

Step 9. Post-Upgrade: Review Data Changes

1. Review warnings/action items generated during upgrade
    - By default, these will be stored in `"C:\ProgramData\Ed-Fi-ODS-Migration\Migration.log"`
    - This message list will contain action items that my require your attention
2. Review your upgraded ODS.
    - A schema named `[v2_to_v3_deprecated]` will be created to store a copy of major objects that were dropped or altered significantly during upgrade, including suite 2 descriptor/type references. If desired, this entire schema is safe to delete once you are positive that the data contained will no longer be needed.

Some data elements that were part of the suite 2 model are either no longer a part of the v3 model, or may be altered to meet the upgraded schema requirements. The following chart summarizes a list of major items that will be altered or dropped during upgrade

Objects dropped during upgrade

What	Where to find it after upgrade
Dropped tables no longer part of the v3 model: • Tables capturing Assessment Family data • Some PostSecondaryInstitution related references no longer used • suite 2 Ed-Fi Descriptors no longer part of the suite 3 model	Tables will be copied to schema `[v2_to_v3_deprecated]` if they are not empty at upgrade time
References to Ed-Fi `*Type` tables that were contained in suite 2 descriptors. Types are no longer a part of the suite 3 model.	The table `[v2_to_v3_deprecated].[TypeToDescriptorAssociationBackup]` will contain a list of deprecated types (`[CodeValue]` only) that were associated with Descriptors at the time of upgrade.
Characteristic data that was stored directly on `[edfi].[Student]` incompatible with the new model because an `[StudentEducationOrganizationAssociation]` reference is now required beginning in v3 . This includes economic disadvantaged status, school food services eligibility, and displacement status.	For all students that contain non-null data in the related columns on `[edfi].[Student]` , an entry will be generated in `[v2_to_v3_deprecated].[Student].`

Notes on key important items altered during upgrade

Affected Objects	Change Description	Why
Types, Descriptors	• Namespace changed from suite 2 format (`http://{edorg}/Descriptor/{DescriptorName}.xml`) to suite 3 format (`uri://{edorg}/{DescriptorName}`)	Significant type/descriptor changes have occurred between suite 2 and suite 3 • Beginning in suite 3, type tables are no longer part of the

	• Rare: In the event that duplicate descriptors associated with the same organization are encountered, an integer will be added to the [CodeValue] for uniqueness • New suite 3 descriptors inserted during upgrade with namespace specified at upgrade time • In the event that a Type table was modified from the default, these changed types will be converted into new descriptors as well Note: The state of the descriptor table with old namespaces is copied to `[v2_to_v3_deprecated].[Descriptor]` at upgrade time	model. These objects will be replaced with several Descriptors new to suite 3. • Namespaces are now required, and must be the new v3 format exactly. The upgrade utility will do the work to convert valid suite 2 namespaces to the new suite 3 format for you. • Rare: The schema requires that each combination of [CodeValue] + [Namespace] be unique for each Descriptor, which could result in potential duplicates in some cases where duplicate suite 2 [CodeValue] entries exist.
`[edfi].[Section]:[UniqueSectionCode] (suite 2)`	• The suite 2 `[UniqueSectionCode]` is used to create the new suite 3 `[SectionIdentifier]` • For sections that are considered duplicates under the new suite 3 model, the new `[SectionIdentifier]` will contain the old `[UniqueSectionCode]` plus an integer that is automatically appended during upgrade.	In the older suite 2 model, the primary key on `[edfi].[Section]` permits a `[UniqueSectionCode]` can be associated with the same School/Session/Course multiple times. As of suite 3, the key has been simplified. A `[SectionIdentifier]` can only be associated with the same School/Session/Course once.
`[edfi].[Session]:[SessionName]`	• In the event that there are two sessions with the same `[SessionName]` are associated with the same school and year, the new `[SessionName]` after upgrading will include the name of the associated term.	In the older suite 2 model, the Session table used `[TermDescriptorId]` in the key. The suite 3 key replaces `[TermDescriptorId]` with `[SessionName]`

Ed-Fi ODS Migration Tool Parameter Reference

Parameter	Description	Example	Required?
`--Database`	Database Connection String	`--Database "Data Source=YOUR\SQLSERVER;Initial Catalog=Your_EdFi_Ods_Database;Integrated Security=True"`	**Yes**
`--DescriptorNamespace`	Descriptor Namespace prefix to be used for new and upgraded descriptors. Namespace must be provided in suite 3 format as follows: `uri://[education_organization_here]` Valid characters for an education organization name: alphanumeric and `$-_.+!*'(),` Script Usage: Provided string value will be escaped and substituted directly in applicable sql where the `$DescriptorNamespace$` variable is used	`--DescriptorNamespace "uri://ed-fi.org"`	**Yes**
`--CredentialNamespace`	Namespace prefix to be used for all new Credential records. Namespace must be provided in suite 3 format as follows: `uri://[education_organization_here]` Valid characters for an education organization name: alphanumeric and `$-_.+!*'(),` Script Usage: Provided string value will be escaped and substituted directly in applicable SQL where the `$CredentialNamespace$` variable is used.	`--CredentialNamespace "uri://ed-fi.org"`	**Yes, if table edfi.StaffCredential has data** Optional if table edfi.StaffCredential is empty
`--CalendarCo`	Path to calendar configuration, which must be accessible from your sql server.		**Single-Year ODS: No** (unless prompted by the upgrade tool)

nfigFilePath	Script Usage: Provided string value will be substituted directly in dynamic SQL where the `$CalendarConfigFilePath$` variable is used. The `'` character is not permitted by the upgrade utility for this value.	`--CalendarConfigFilePath "C:\PATH\TO\YOUR\CALENDAR_CONFIG.csv"`	**Multi-Year ODS: Yes**
`--DescriptorXMLDirectoryPath`	Path to directory containing 3.1 descriptors for import, which must be accessible from your sql server Script Usage: Provided string value will be substituted directly in dynamic SQL where the `$DescriptorXMLImportDirectoryPath$` variable is used. The `'` character is not permitted by the upgrade utility for this value.	`--DescriptorXMLDirectoryPath "C:\PATH\TO\YOUR\DESCRIPTOR\XML"`	**Local Upgrade**: **No** (applicable to most cases) **Remote Upgrade**: **Yes** Used if the Descriptor XML directory has been moved to a different location accessible to your sql server
`--BypassExtensionValidationCheck`	Permits the migration tool to make changes if extensions or external schema dependencies have been found	`--BypassExtensionValidationCheck`	**Extended ODS: Yes** This includes any dataset with an extension schema or foreign keys pointing to the Ed-Fi schema. **Others**: **No**
`--CompatibilityCheckOnly`	Perform a dry run for testing only to check ODS compatibility without making additional changes. **The database will not be upgraded.**	`--CompatibilityCheckOnly`	**No.** This is an optional feature.
`--Timeout`	SQL command query timeout, in seconds.	`--Timeout 1200`	**No.** (Useful mainly for development and testing purposes)
`--ScriptPath`	Path to the location of the SQL scripts to apply for upgrade, if they have been moved	`--ScriptPath "C:\PATH\TO\YOUR\MIGRATION_SCRIPTS"`	**No.** (Only needed if scripts have been moved from the default location. Useful mainly for development and testing purposes)
--FromVersion	The ODS/API version you are starting from.	--FromVersion "3.1.1"	**No.** Migration utility will detect your staring point.
--ToVersion	The ODS/API version that you would like upgrade to.	--ToVersion "5.0.0"	**No.** Migration utility will take you to the latest version by default.

Usage Troubleshooting Guide

The below section provides additional guidance for many common compatibility issues that can be encountered during the upgrade process.

Error received during upgrade	Explanation	How to fix it
Action Required: Unable to proceed with migration because the BypassExtensionValidationCheck option is disabled ...	An external dependency on the edfi schema has been found. As a courtesy, the migration tool will not proceed with the upgrade process without your permission. **Common Examples:** • You have a table on the extension schema • An object on **any schema other than `edfi`** has a foreign key that depends directly on data on the `edfi` schema **Why:** This notification is intended to bring extension items to your attention that will require manual handling. All primary keys and indexes on the `edfi` schema are dropped during upgrade. The existence of any external constraint that depends on these objects will result in an unhandled sql exception. (For details, see the troubleshooting section: `"SqlException : The constraint '{CONSTRAINT_NAME_HERE}' is being referenced by table '{TABLE_NAME_HERE}', foreign key constraint '{FORIEGN_KEY_NAME_HERE}'"`	After reviewing the data and dependencies on your extension tables, add the `--BypassExtensionValidationCheck` option at the command line. This will give the migration tool permission to proceed even if there are extension items present. Please review Step 8. Write Custom Migration Scripts for Extensions (above) before proceeding.
Action Required: edfi.StaffCredential ...	The column **`StateOfIssueStateAbbreviationTypeId`** must be non-null for all records. This value will become part of a new primary key on the 3.1 schema.	Add a [StateOfIssueStateAbbreviationTypeId] for all records in [edfi].[StaffCredential]. This is the abbreviation for the name of the state (within the United States) or extra-state jurisdiction in which a license or credential was issued. ✅ The table is compatible for upgrade if the below query returns 0 results. **Expand to see code**

		SELECT * FROM [edfi]. [StaffCredential] WHERE [StateOfIssueStateAbbrevia tionTypeId] IS NULL
Action Required - An EducationOrganizationId must be resolvable for every student in the following table(s) for compatibility with the upgraded schema starting in version 3.0: **(Provided list of tables includes one or more of the following):** • edfi. StudentProgramParticipa tion • edfi. StudentCharacteristic • edfi.StudentIndicator • edfi. StudentLearningStyle • edfi.StudentAddress • edfi. StudentIdentificationCode • edfi. StudentElectronicMail • edfi. StudentInternationalAddr ess • edfi.StudentLanguage • edfi.StudentRace • edfi.StudentDisability • edfi.StudentTelephone • edfi. PostSecondaryEventPost SecondaryInstitution	The upgrade utility must be able to locate an **[EducationOrganizationId]** for every student with data in the listed tables to proceed. Beginning in v3.1, the schema structure now requires that these student information items be defined separately for each associated EducationOrganization rather than simply linking them to a student.	The easiest way to meet this requirement is to ensure that every student has a corresponding record in **[edfi].[StudentSchoolAssociation]** or **[edfi]. [StudentEducationOrganizationAss ociation].** The upgrade tool will use this information to handle the rest of the data conversion tasks for you.
Action Required: edfi. Assessment ...	All assessments must have a [Namespace] set. (This data may be found in [edfi].[Assessment] or [edfi].[AssessmentFamily]). In suite 3, the schema required that this column be non-null.	Add a [Namespace] for all assessment records. ✅ The table is compatible for upgrade if the below query returns 0 results **Expand to see code** SELECT * FROM [edfi].[Assessment] a LEFT JOIN [edfi]. [AssessmentFamily] f ON a. [AssessmentFamilyTitle] = f.[AssessmentFamilyTitle] WHERE COALESCE(a. [Namespace], f. [Namespace]) IS NULL
Action Required: edfi. OpenStaffPosition ...	There may be no two duplicate **`RequisitionNumber`** entries for the same education organization. This is uniqueness if required for the upgraded primary key on this table.	Update the **RequisitionNumber** values on **edfi.OpenStaffPosition.** Ensure that the same value is not used twice for the the same education organization.

		✅ The table is compatible for upgrade if the below query returns 0 results. **Expand to see code** `SELECT [EducationOrganizationId], [RequisitionNumber], COUNT ([RequisitionNumber]) AS [NumberOfMatchingRecords] FROM [edfi]. [OpenStaffPosition] GROUP BY [EducationOrganizationId], [RequisitionNumber] HAVING COUNT ([RequisitionNumber]) > 1`
Action Required: edfi. RestraintEvent ...	There may be no two duplicate **`RestraintEventIdentifier`** values for the same student at the same school. This is uniqueness if required for the upgraded primary key on this table.	Update the **`RestraintEventIdentifier`** values on **edfi.RestraintEvent**. Ensure that the same **`RestraintEventIdentifier`** is not reused for the same student at the same school. ✅ The table is compatible for upgrade if the below query returns 0 results. **Expand to see code** `SELECT [RestraintEventIdentifier], [SchoolId], [StudentUSI], COUNT ([RestraintEventIdentifier]) AS [NumberOfDuplicateRecords] FROM [edfi]. [RestraintEvent] GROUP BY [RestraintEventIdentifier], [SchoolId], [StudentUSI] HAVING COUNT ([RestraintEventIdentifier]) > 1`
Action Required: edfi. GradingPeriod ...	There may be no two duplicate **`PeriodSequence`** values for the same school during the same grading period. Additionally, if prompted by the upgrade tool, all **`PeriodSequence`** values must be non-null. This compatibility requirement is a result of a primary key change between suite 2 and suite 3 • **Old suite 2 Primary Key:** GradingPeriodDescriptorId, SchoolId, BeginDate • **New suite 3 Primary Key:** GradingPeriodDescriptorId, SchoolId, **PeriodSequence (new)**, **SchoolYear (new)**. *(BeginDate is removed.)*	Ensure that there are no two records with the same **SchoolId**, **GradingPeriodDescriptorId**, **PeriodSequence**, and **SchoolYear**. • Note: The `edfi.GradingPeriod` new `SchoolYear` column is derived from the old suite 2 `BeginDate` value. The SchoolYear from your calendar configuration will be used. • If there are multiple records with the same **SchoolId**, **GradingPeriodDescriptorId**, **PeriodSequence**, and **SchoolYear** you must make changes to ensure that all records are unique. • *Tip: Depending on the needs of your*

		organization, you may decide to remove unneeded conflicting records, update the edfi .GradingPeriod.PeriodSequence values, or assign a different value to edfi.GradingPeriod. PeriodSequence. GradingPeriodDescriptorId. ✅ Tip: You can be certain that the table is compatible for upgrade if the below query returns 0 results. **Version 1 - Simplified Calendar:** **Expand to see code** `SELECT [SchoolId], [GradingPeriodDescriptorId], [PeriodSequence], COUNT([PeriodSequence]) AS [NumberOfDuplicateRecords] FROM [edfi].[GradingPeriod] GROUP BY [SchoolId], [GradingPeriodDescriptorId], [PeriodSequence] HAVING COUNT([PeriodSequence]) > 1` (Requirements may be less strict than noted in the above query for some some multi-year Calendars. See the compatibility check script in the 01 Bootstrap directory for exact technical requirements.)
Action Required: edfi.DisciplineActionDisciplineIncident ...	Every record in [edfi].[DisciplineActionDisciplineIncident] must have a corresponding record in [edfi].[StudentDisciplineIncidentAssociation] with the same [StudentUSI], [SchoolId], and [IncidentIdentifier]. The V3 schema no longer allowed discipline action records with students that are not associated with the discipline incident. A foreign key will be added to the new schema enforcing this.	• Remove any records in [edfi].[DisciplineActionDisciplineIncident] that involve a student not associated with the incident. • For students that are involved in the incident, add a corresponding record to [edfi].[StudentDisciplineIncidentAssociation] with the same [StudentUSI], [SchoolId], and [IncidentIdentifier]. ✅ The table is compatible for upgrade if the below query returns 0 results **Expand to see code** `SELECT d.* FROM [edfi].[DisciplineActionDisciplineIncident] d LEFT JOIN [edfi].[StudentDisciplineIncidentAssociation] s ON s.[StudentUSI] = d.[StudentUSI] AND s.[SchoolId] = d.[SchoolId] AND s.[IncidentIdentifier] = d.[IncidentIdentifier]`

		`WHERE s.[StudentUSI] IS NULL`

Calendar configuration file error - *various similar messages may appear that mention a* ***table name*** *and a list of* ***school ids****.*
 	
Example error:Found {#} date ranges in **[edfi].[** `Table name will vary: Session, CalendarDate, GradingPeriod` **]** which did not fall within the dates specified in the calendar configuration. The top 10 affected schools will be listed
	The calendar configuration file contains the start date and end date for each school year. To support the new calendar features in V3, the migration tool uses this configuration file to assign a SchoolYear to all CalendarDate related entries in the database.
	
	There are several variations of this type of error which all have a similar meaning. The migration tool found date records in the specified table that could not be assigned a school year based on the BeginDate and EndDate information provided.
		Either the calendar configuration file will need to be edited, or data in the specified table will need to be modified.
		
		1. Make note of the list of **School Ids** and the **Table** name provided in the error message.
		2. Open your calendar configuration file. Find the line corresponding to each **School Id**. Ensure that the `BeginDate` and `EndDate` for each `SchoolYear` is accurate.
		   • If a **School Id** is missing from the calendar configuration file, be sure to add it.
		3. Check the date information in the specified table for accuracy.
		   • Carefully check the ODS records in **table** specified by the error message for the specified **School Id**.
		   • Make sure there are no date entries that fall outside the school year information you have provided in the calendar configuration.
		   • If the table already has `SchoolYear` data (such as `edfi.Session` or `edfi.GradebookEntry`), make sure the `SchoolYear` data is accurate.
		
		Example:
		
		Example Query: [edfi].[Calendar Date]
		
		`SELECT * FROM [edfi].[CalendarDate] WHERE [SchoolId] = {School_Id_from_error_message} AND ( [Date] < {BeginDate_from_calendar_configuration_file} OR [Date] > {EndDate_from_calendar_configuration_file} )`

SqlException: The constraint '{CONSTRAINT_NAME_HERE}' is being referenced by	This type of unhandled SQL exception occurs when the migration process tries to alter an item that is being referenced by an external object, such as a foreign key on another table, or schema-bound view.	Make sure that you have dropped ALL foreign keys and views from other schemas that have a dependency on the `edfi` schema.

table '{TABLE_NAME_HERE}', foreign key constraint '{FORIEGN_KEY_NAME_HERE}' or similar: The object '{OBJECT_NAME_HERE}' is dependent on column '{COLUMN_NAME_HERE}'.	**Common causes** • The `--BypassExtensionValidationCheck` option was enabled, and the EdFi Migration script tried to drop an index that is being referenced by a foreign key on another schema. • There is a schema-bound view with a direct dependency on a table that is being modified.	This must be done by adding custom migration scripts for your ODS, or by dropping these items by hand. *Any external dependencies present will result in a SQL exception.* You do not need to drop any constraints on the edfi schema itself: This is handled automatically for you. For tips on locating foreign key dependencies quickly, see the query in Step 8b. above.

A data validation failure was encountered on destination object {*table name here*} ...	Data was modified in a location that was not expected to change. A validation check from script directory `Scripts/02 Setup/{version}/## Source Validation Check` has failed.	This state is triggered if certain records are modified in the middle of migration that the upgrade utility expected would remain unchanged. The upgrade will be halted for you as a precaution to prevent unintended data loss.
		
		Common Causes:
		
		• Updates were made (either by hand or other means) in the middle of the migration process after the data compatibility checks passed.
		• Custom migration scripts performed an insert/update /delete on an object that the migration utility expects to remain unchanged by default.
		
		If you are testing potential updates to the suite 2 ODS data by hand (data only / no custom scripts):
		
		• For testing of **data** updates only: Simply restore the target suite 2 ODS and make the desired inserts/updates/deletes before launching the utility.
		  • *It is recommended to create your own custom migration scripts if testing more complex operations*
		• You should ensure that all suite 2 data updates have been applied before the data compatibility check passes.
		
		If you are writing your own custom scripts:
		
		• Check the output of the table mentioned in the error message carefully and ensure that the detected suite 3 change is as intended.
		• If the error references a table /column that you know is intended to change during upgrade, you may optionally remove the data validation check.
		
		  • These items can be found in directory `Scripts/02 Setup/{version}/## Source Validation Check`
		  • *It is not recommended to modify the edfi schema, as this may make upgrade tasks difficult in the future. By convention, custom tables are normally found in an outside location, such as the extension schema.*

		If your suite 2 Ed-Fi ODS schema is unmodified, and you are not making edits to the current scripting or data, this error should not occur during normal operation. • Try the upgrade process again, and if the same error occurs, you may have encountered a bug.