Extending the SDG - Student Transportation Example (v2.x)

Contents

This section walks through extending the SDG to produce a new XML Interchange file. **This example produces XML aligned with the Ed-Fi Technical Suite 2 and produces XML conformant to the Data Standard v2.x specifications.** If you need to generate sample data for Technical Suite 3, see this article instead.

The example herein is based on the How To: Extend the Ed-Fi ODS / API - Student Transportation Example. It's assumed you already have access to the SDG code, have forked the SDG repository (master-v2 or master-v22 branch depending on the targeted version of the Data Standard), and have cloned the repo to your development machine. References below use Ed-Fi Data Standard v2.2 and its associated master-v22 branch, but use master-v2 if targeting Ed-Fi Data Standard v2.0.

The general steps for this process are:

Detail on each step follows.

> **Note: Repo Location**
>
> For the purposes of this walkthrough, we'll use paths relative to the root of your repo. For example, if you've saved code to *C:\Code\Ed-Fi-SDG*, when you see a path of the form *Application\EdFi.SampleDataGenerator.Core*, that refers to *C:\Code\Ed-Fi-SDG\Application\EdFi.SampleDataGenerator.Core*

Step 1. Import the Extended XSDs into the SDG Project

Like the ODS itself, extending the SDG starts with the extension XSD files for your use case. Import your XSDs as follows:

1. Copy the contents of *EXTENSION-Ed-Fi-Core.xsd* and *EXTENSION-Interchange-StudentEnrollment.xsd* from the Student Transportation example to Application\EdFi.SampleDataGenerator.Core\Entities\Schema\v22\Extensions. For convenience, these files have also been copied SampleExtensions\Transportation in the root of the repository.
2. Modify *EXTENSION-Ed-Fi-Core.xsd* with correct relative path for schemaLocation:

EXTENSION-Ed-Fi-Core.xsd

```
<xs:include schemaLocation="..\Ed-Fi-Core.xsd"/>
```

3. Copy *Application\EdFi.SampleDataGenerator.Core\Entities\Ed-Fi-Ods-Xsd-Generator-v22.xsd* to *EXTENSION-Ed-Fi-Ods-Xsd-Generator-v22.xsd*
4. Modify *EXTENSION-Ed-Fi-Ods-Xsd-Generator-v22xsd* to include your extension. A sample extension is provided in SampleExtensions\Transportation\Entities\Schema\v30\Extensions in the root of the repository.

EXTENSION-Ed-Fi-Ods-Xsd-Generator-v22.xsd

```
<schema>.\Schema\v22\Extensions\EXTENSION-Interchange-
StudentEnrollment.xsd</schema>
```

Step 2. Run Xsd.exe to Produce C# Entities that Can Be Populated

For this step, you MUST run from a Visual Studio Command prompt or otherwise have Xsd.exe available in your PATH.

Run Xsd.exe via Ed-Fi-Ods-Xsd-Generator.bat

```
C:\Code\Ed-Fi-SDG\Application\EdFi.SampleDataGenerator.Core\Entities>Ed-Fi-
Ods-Xsd-Generator.bat EXTENSION-Ed-Fi-Ods-Xsd-Generator-v22.xsd
```

Note the new changes introduce into Application\EdFi.SampleDataGenerator.Core\Entities\EdFiEntities.cs.

Step 3. Make the SDG Aware of Your New Extension Type(s)

In this example we're adding new entities to an existing Interchange - namely StudentEnrollment. We need to register the new StudentTransportation type as part of the StudentEnrollment Interchange. This is accomplished by editing the contents of *Application\EdFi.SampleDataGenerator.Core\DataGeneration\InterchangeEntities\StudentEnrollmentEntity.Extensions.cs*.

Application\EdFi.SampleDataGenerator.

Core\DataGeneration\InterchangeEntities\StudentEnrollmentEntity.Extensions.cs

```
using EdFi.SampleDataGenerator.Core.Entities;
namespace EdFi.SampleDataGenerator.Core.DataGeneration.InterchangeEntities
{
        public partial class StudentEnrollmentEntity
        {
                public static readonly StudentEnrollmentEntity
StudentTransportation = new StudentEnrollmentEntity(typeof
(EXTENSIONStudentTransportation));
        }
}
```

Step 4. Add Properties to Context Type(s)

Generators in the SDG are passed a Context object which is progressively populated with data.

Once all generators have been run, the given Context is a fully formed ODS record ready for output. Therefore, when adding a new type as we are doing in this example, the Context must be updated with a place for *StudentTransportation* data to be populated by a generator. Since we're adding on to the *StudentEnrollment* Interchange, we'll modify the Context type already built into the core SDG by editing *Application\EdFi.SampleDataGenerator.Core\Serialization\Output\Interchanges\StudentEnrollmentData.Extensions.cs*.

Application\EdFi.SampleDataGenerator.

Core\Serialization\Output\Interchanges\StudentEnrollmentData.Extensions.cs

```
using System.Collections.Generic;
using EdFi.SampleDataGenerator.Core.Entities;

namespace EdFi.SampleDataGenerator.Core.Serialization.Output.Interchanges
{
    public partial class StudentEnrollmentData
    {
        public List<EXTENSIONStudentTransportation> StudentTransportations
{ get; set; } = new List<EXTENSIONStudentTransportation>();
    }
}
```

Step 5. Write Generators to Populate Extension Data

Now we're ready to create a generator to populate *StudentTransportation* data. There are a few properties required by the inheritance chain:

- A 1-argument constructor that takes an *IRandomNumberGenerator* and passes that object on to *base()*.

- *GeneratesEntity.* This will match up with the type you created above in the Make the SDG Aware of Your New Extension Type(s) section.
- *DependsOnEntities.* Here you'll define any entities your generator needs to be populated prior to running.

You'll place any business logic for generating one-time data into *GenerateCore*; logic for data that should be generated per data period should be placed in *GenerateAdditiveData*.

Application\EdFi.SampleDataGenerator.

Core\DataGeneration\Generators\StudentEnrollment\StudentTransportationEntityGenerator.cs

```
using EdFi.SampleDataGenerator.Core.DataGeneration.Common;
using EdFi.SampleDataGenerator.Core.DataGeneration.Common.Dependencies;
using EdFi.SampleDataGenerator.Core.DataGeneration.Common.Entity;
using EdFi.SampleDataGenerator.Core.DataGeneration.InterchangeEntities;
using EdFi.SampleDataGenerator.Core.Entities;
using EdFi.SampleDataGenerator.Core.Helpers;

namespace EdFi.SampleDataGenerator.Core.DataGeneration.Generators.
StudentEnrollment
{
    public class StudentTransportationEntityGenerator :
StudentEnrollmentEntityGenerator
    {
        public StudentTransportationEntityGenerator(IRandomNumberGenerator
randomNumberGenerator) : base(randomNumberGenerator)
        {
        }

        private const double RidesBusChance = 0.55;

        public override IEntity GeneratesEntity => StudentEnrollmentEntity.
StudentTransportation;
        public override IEntity[] DependsOnEntities => EntityDependencies.
Create(StudentEntity.Student);

        protected override void GenerateCore(StudentDataGeneratorContext
context)
        {
            var ridesBus = RandomNumberGenerator.GetValueWithProbability
(RidesBusChance, true, false);

            if (ridesBus)
            {
                context
                    .GeneratedStudentData
                    .StudentEnrollmentData
                    .StudentTransportations
                    .Add(new EXTENSIONStudentTransportation
                    {
                        StudentReference = context.Student.
GetStudentReference(),
                        SchoolReference = Configuration.SchoolProfile.
GetSchoolReference(),
                        AMBusNumber = GenerateBusNumber(),
                        PMBusNumber = GenerateBusNumber(),
                        EstimatedMilesFromSchool = GenerateEstimatedMiles()
                    });
            }
        }

        private string GenerateBusNumber()
        {
            return RandomNumberGenerator.Generate(100, 999999).ToString();
        }

        private decimal GenerateEstimatedMiles()
        {
            return decimal.Parse(
                RandomNumberGenerator.Generate(0, 900)
                + "." +
                RandomNumberGenerator.Generate(10, 99));
        }
    }
}
```

Step 6. Run Unit Tests

There are some important conventions that MUST be followed in any extension code you create, especially when it comes to setting up your new entities in the Context. Note that in the code snippet above we chose to initialize the StudentTransportations collection with a default empty list. This is a requirement and is enforced when you run the Unit tests for the project. You should ALWAYS run unit tests after adding new code to ensure that code conforms to conventions. Otherwise, you may get runtime errors.

Step 7. Run the SDG to Produce Your Data

With your new generator(s) in place, run the SDG and check your output. Here's a standard set of command line parameters used by the SDG development team. You can place these parameters in the EdFi.SampleDataGenerator.Console project command line parameters configuration to enable successful F5 run behavior from Visual Studio.

Command line parameters
`-c Samples\v22\SampleDataGenerator\SampleConfig.xml -d Samples\v22\SampleDataGenerator\DataFiles\ -o C:\Temp\Northridge\Output\ -AllowOverwrite`