|  |  |
| --- | --- |
| 1 | Знакомство со службой GPSD в Linux среде. Настройка окружения. Visual Studio (Rider), PgAdmin. Подключение к серверу. |
| 2 | Знакомство со структурой Json данных. Работа с Json типовым набором библиотек C#, System.Text.Json. Работа с Json с использованием библиотеки Newton.Json. Сериализация / десериализация. |
| 3 | Разработка структур данных в среде C# для десериализации данных. Создание тестов в NUnit. |
| 4 | Полная загрузка и десериализация всех данных с использованием шаблона *Фабрика*. |
| 5 | Изучение и разработка систем параллельного выполнения заданий в среде C#. Шаблон *Наблюдатель* |
| 6 | Реализация данного шаблона на примере загрузки данных из GPSD сервиса. |

**Знакомство со службой GPSD в Linux среде.**

1. Знакомство с приложение Gps test  
<https://play.google.com/store/apps/details?id=com.chartcross.gpstest&hl=ru&gl=US>  
  
Установка. Поиск спутников.

2. Формат передачи данных службой Gpsd. Используется тескстовый формат (собственный, json). При вызове сервиса **gpsmon** , получаем

|  |
| --- |
|  |

**Знакомство со структурой Json данных**

Для начала, добавим в проект 2 файла с примером трассировок Gpsd сервиса.

|  |
| --- |
| gpsd\_exists\_data.json |
| gpsd\_not\_exists\_data.json |

Файл gpsd\_not\_exists\_data.json – содержит трассировку без обнаруженных спутников. Файл gpsd\_exists\_data.json – содержит трассировку полную но с “разогревом”.

1 Парсинг текстовых данных.

2 Сериализация / Десериализация Json данных.  
<https://learn.microsoft.com/ru-ru/dotnet/standard/serialization/system-text-json/how-to?pivots=dotnet-6-0>

Первый этап: Создаем общую структуру в виде:

|  |  |
| --- | --- |
| Объект | Назначение |
| IGpsDataModel.cs | Общий интерфейс для каждой модели данных |
| DataModelType | Перечисления со списком типов данных |
| GpsdDataModel | Реализация интерфейса |
| GpsdDataManager | Основной класс для работы с сервисов Gpsd и набором данных. Обработка данных. |

Детали:  
Класс реализации имеет два поля: Class, Type Свойство Type имеет аттрибут – только чтение. Это свойство формируется автоматически при укказании поля Class.

|  |
| --- |
| /// <summary> /// Наименование класса модели /// </summary> [JsonPropertyName("class")] public string Class { get => \_class;  set  {  \_class = value ?? throw new ArgumentNullException(nameof(Class));  Enum.TryParse<DataModelType>(\_class, true, out \_modelType);  } } |

Добавим метод в класс GpsdDataManager

|  |
| --- |
| /// <summary>  /// Сконвертировать строковые данные в модель типа <see cref="IGpsDataModel"/>  /// </summary>  /// <param name="data"> Строковые данные </param>  /// <returns></returns>  /// <exception cref="ArgumentNullException"></exception>  public IGpsDataModel ConvertToModel(string data)  {  if (string.IsNullOrEmpty(data))  throw new ArgumentNullException(nameof(data));   var result = JsonSerializer.Deserialize<GpsdDataModel>(data)  ?? throw new InvalidDataException(  "Невозможно сконвертировать входные данные в тип IGpsDataModel!");  return result;  } } |

Далее, добавим автоматический тест и выполним его. Пример:

|  |
| --- |
| [TestCase(@"{""class"":""TPV"",""device"":""/dev/ttyS3"",""mode"":1,""time"":""1980-01-06T00:22:47.868Z"",""ept"":0.005}")] public void Check\_ConvertToModel(string data) {  // Подготовка    // Дейтсвие  try  {  var result = \_manager.ConvertToModel(data);  Assert.IsNotNull(result.Class);   var item = result as GpsdDataModel;  Assert.IsNotNull(item);  Assert.That(item?.ModelType.ToString(), Is.EqualTo(item?.Class ?? ""));  }  catch (InvalidDataException ex)  {  Console.WriteLine(ex);  Assert.Fail();  } } |

Итоги: Создали общую структуру. Научились работать с сериализацией.

Задание:   
1 Создать все модели данных для структур типа: GST, TPV, WATCH, DEVICES, VERSION, SKY с использованием наследования

2 Сделать для всех моделей автоматические теста.

Добавим следующие модели

|  |  |
| --- | --- |
| GpsdVersionModel |  |
| GpsdSkySatelliteModel |  |
| GpsdSkyModel |  |
|  |  |

Модель GpsdSkyModel включает в себя табличную часть GpsdSkySatelliteModel с информацией о нахождении спутников. Свяжем модели.

|  |
| --- |
| [JsonPropertyName("satellites")] public IEnumerable<GpsdSkySatelliteModel> Satellites { get; set; } |

Доработаем метод ConvertToModel. Свяжем с Generic типом T

|  |
| --- |
| /// <summary> /// Сконвертировать строковые данные в модель типа <see cref="IGpsDataModel"/> /// </summary> /// <param name="data"> Строковые данные </param> /// <returns></returns> /// <exception cref="ArgumentNullException"></exception> public T ConvertToModel<T>(string data) where T : class, IGpsDataModel {  if (string.IsNullOrEmpty(data))  throw new ArgumentNullException(nameof(data));   var result = JsonSerializer.Deserialize<T>(data)  ?? throw new InvalidDataException(  "Невозможно сконвертировать входные данные в тип IGpsDataModel!");  return result; } |

Так же, изменим автотест.

|  |
| --- |
| var result = \_manager.ConvertToModel<GpsdDataModel>(data); |

И добавим два новых автотеста для обработки моделей SKY, VERSION по аналогии

Итого: Преобразовали механизм десериализацию . Сделали его более универсальным. Доработали автотесты.

Задание:

Автотест

public void Check\_ConvertToModel\_GpsdSkyModel(string data)

Доработать. Сделать проверку количество спутников.

***Полная загрузка и десериализация всех данных с использованием шаблона Фабрика.***

|  |
| --- |
| https://en.wikipedia.org/wiki/Factory\_(object-oriented\_programming) |
| https://habr.com/ru/post/465835/ |

Для начала создадим специальный класс **GpsdDataFactory**

Назначение: Создавать требуемый класс модели в зависимости от типа перечисления DataModelType

1 Создадим интерфейс для фабрики

|  |
| --- |
| /// <summary> /// Интерфейс - фабрика  /// </summary> public interface IGpsdDataFactory {  /// <summary>  /// Создать модель  /// </summary>  /// <param name="type"></param>  /// <returns></returns>  public IGpsDataModel? CreateModel(DataModelType type);   /// <summary>  /// Определить тип модели  /// </summary>  /// <param name="type"></param>  /// <returns></returns>  public Type GetTypeOfModel(DataModelType type); } |

2 Далее, добавим реализацию фабрики. В качестве механизма реализации включим в класс словарь Idictionary<DataModelType, Type>

|  |
| --- |
| /// <summary> /// Реализация интерфейса <see cref="IGpsdDataFactory"/> /// </summary> public class GpsdDataFactory: IGpsdDataFactory {  // Карта связей  private IDictionary<DataModelType, Type> \_templates = new Dictionary<DataModelType, Type>()  {  { DataModelType.SKY, typeof(GpsdSkyModel) },  { DataModelType.VERSION, typeof(GpsdVersionModel) }  };   /// <summary>  /// Создать объект треебуемого типа  /// </summary>  /// <param name="type"></param>  /// <returns></returns>  public IGpsDataModel? CreateModel(DataModelType type)  => Activator.CreateInstance(this.GetTypeOfModel(type)) as IGpsDataModel;    /// <summary>  /// Получить требуемый тип данных по значению перечисления <see cref="DataModelType"/>  /// </summary>  /// <param name="type"></param>  /// <returns></returns>  /// <exception cref="InvalidEnumArgumentException"></exception>  public Type GetTypeOfModel(DataModelType type)  => \_templates.ContainsKey(type)  ? \_templates[type]  : throw new InvalidEnumArgumentException(  "Некорректно переданы параметры! Для указанного перечисления нет реализации!");  } |

3. Добавим автоматический тест для проверки работы фабрики. Для этого создадим отдельный тестовый класс.

|  |
| --- |
| /// <summary> /// Проверить получение типа по перечислению. Верные данные. /// </summary> /// <param name="type"></param> [Test] [TestCase(DataModelType.SKY, typeof(GpsdSkyModel))] [TestCase(DataModelType.VERSION, typeof(GpsdVersionModel))] public void Check\_GetType\_Try(DataModelType typeOfModel, Type typeAssert) {  // Подготовка  var factory = new GpsdDataFactory();    // Действие  var result = factory.GetTypeOfModel(typeOfModel);    // Проверки  Assert.IsNotNull(result);  Assert.That(typeAssert, Is.EqualTo(result)); } |

Задание: Создать самостоятельно тест для проверки работы фабрики с некорректными аргументами.

***Изучение и разработка систем параллельного выполнения заданий в среде C#. Шаблон Наблюдатель***