# Лабораторная работа №4Сериализация и десериализация данных

**Цель работы**: Изучение механизмов сохранения и восстановления состояния объектов. Изучение особенностей сериализации и десериализации данных в разных форматах.

## Задание

На основе проектов, входящих в состав лабораторной работы № 2 и 3, дополнить классы, соответствующие своему варианту предметной области, средствами сериализации в файл и десериализации из файла. С помощью стандартных диалогов *OpenFileDialog* и *SaveFileDialog* организовать интерфейс пользователя для указания имени и типа файла. Основные требования к работе представлены в таблице 4.1.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Требования к работе | Баллы | Оценка |
|  | Сериализация и десериализация в формате *JSON*/*Binary*/*XML* (по баллу за каждый формат) | 1/1/1 |  |
|  | Использования стандартных компонентов *OpenFileDialog* и *SaveFileDialog* | 1 |  |
|  | Использование ключевого слова *partial*  для разделения кода, который относится к одному классу, но к разным задачам (лабораторным работам) | 1 |  |
|  | *Custom serialization*. Использование *ISerializable* (или *IXmlSerializable*) в качестве наследуемого интерфейса в классах предметной области  | 2 |  |
|  | Использование *callback* механизма в задачах сериализации | 1 |  |

## Порядок выполнения работы

Реализация задач сериализации может быть разделена на две части – создание методов чтения/записи данных и использование классов-атрибутов для определения структуры сериализуемых данных (таблица 4.2).

|  |
| --- |
| Таблица 4.2 |
| **Задача** | **Формат сериализации** |
| ***JSON*** | ***Binary*** | ***XML*** |
| Блокировка сериализации | [IgnoreDataMember] | [NonSerialized] | [XmlIgnore] |
| Управление полями структур | [DataMember] | public/private{ get; set; } | [XmlAttribute] |
| Управление структурами | [DataContract] | [Serializable] | [XmlType] [XmlText] [XmlEnum]  |
| Управление коллекциями | [CollectionDataContract] | [XmlRoot] [XmlElement] |
| Привязка типа к сериализатору | DataContractJsonSerializer(typeof(***TypeName***)) | BinaryFormatter() | XmlSerializer(typeof(***TypeName***)) |
| Запись / Чтение | WriteObject(***file***, ***obj***)ReadObject(***file***) | Serialize(***file***, ***obj***)Deserialize(***file***) | Serialize(***file***, ***obj***)Deserialize(***file***) |

Операции чтение и записи достаточно схожи между собой и, например, для формата *XML* они могут выглядеть следующим образом:

void Save(string filename, ***<сериализуемый класс***> data)

{

 XmlSerializer fmt = new XmlSerializer(typeof(***<сериализуемый класс***>));

 using (FileStream file = new FileStream(filename, FileMode.Create))

 {

 fmt. Serialize(file, data);

 }

}

***<сериализуемый класс>*** Load(string filename)

{

 XmlSerializer fmt = new XmlSerializer(typeof(<***сериализуемый класс***>));

 using (FileStream file = new FileStream(filename, FileMode.Open))

 {

 return (***<сериализуемый класс>***)fmt.Deserialize(file);

 }

}

Необходимо обратить внимание, что для использования сериализации и десериализации в формате *JSON* необходимо наличие в системе *.Net Framework* не ниже версии 4.5 (входит в состав *Visual Studio* 2012 или выше). Кроме этого в рабочий проект необходимо добавить ссылку на сборку (*assembly*) с необходимыми классами (см. рисунок 4.1).



Рисунок 4.1 – Подключение в проект сборки с необходимыми для  *JSON*-сериализации классами