

XmlSerializer

Чтобы создать класс, который можно сериализовать в формат XML, нужно выполнить следующее:

- объявить класс как открытый (public)

- объявить все члены, которые нужно сериализовать, как открытые (public)

- создать конструктор по умолчанию (без параметров)

Условия для XmlSerializer

1-должен иметь конструктор по умолчанию

2-только сериализация открытых членов и типов (public полей или public значений возвращающих свойствами)

3-помечать сериализуемый класс атрибутом Serializer не обязательно

1

Изначально нужно рутовый узел (но можно и не назначать)

XmlRoot – если не применять то имя рутового узла будет имя серилизованного класса

XmlRoot - Переименовывает рутовую ноду.

[XmlRoot("MyButton")]

public class MyClass

XmlAttribute

XML атрибут переименовываем и делаем атрибутом.

[XmlAttribute("SerialID")]

public string ID

XmlElement

XML элемент.

[XmlElement("Pos")]

public **Point** Position

XmlIgnore

Исключаем свойство из процесса сериализации/десериализации.

[XmlIgnore]

public string Password

XmlArray XmlArrayItem

Характеристика массива.

[XmlArray("List")]

// Характеристика каждого элемента массива.

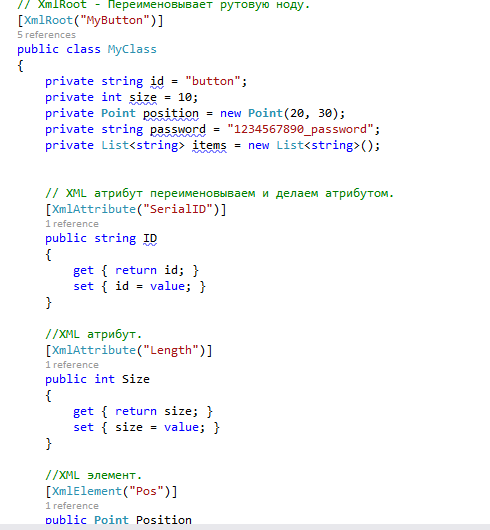
[XmlArrayItem("Element")]

public List<string> Items

{

get { return items; }

ОПИСАНИЕ



2

Serialize

// Тип для Сериализации и Десериализации.

readonly XmlSerializer serializer = new XmlSerializer(typeof(MyClass));

using (var stream = new FileStream("Serialization.xml", FileMode.Create, FileAccess.Write, FileShare.Read))

{

// Сохраняем объект в XML-файле на диске(СЕРИАЛИЗАЦИЯ).

serializer.Serialize(stream, instance1); первым поток, вторым объект для сериализации

this.Text = "Объект сериализован!";

}

ДЕСЕРИАЛИЗАЦИЯ

Deserialize

using (var stream = new FileStream("Serialization.xml", FileMode.Create, FileAccess.Write, FileShare.Read))

{

// Сохраняем объект в XML-файле на диске(СЕРИАЛИЗАЦИЯ).

serializer.Serialize(stream, instance1);

this.Text = "Объект сериализован!";

}

Результат сериализаци



Если не применять атрибуты к свойствам для сериализации, то имена в XML документе примут названия членов класса



BinaryFormatter

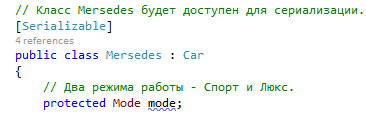
– сериализует абсолютно все поля (состояние класса) в независимости от модификатора доступа

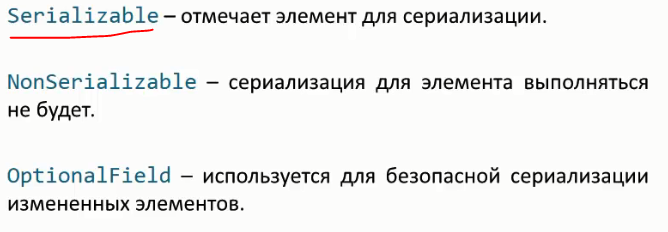
Сериализация в двоичном формате

(если десириализация будет выполняться не на .NET то данная сериализация не подойдет)

Класс который необходимо сериализовать должен быть декорирован атрибутом

[Serializable]





Сериализация

FileStream stream = File.Create("CarData.dat");

// Помещаем объектный граф (для базовых типов) в поток в двоичном формате.

BinaryFormatter formatter = new BinaryFormatter();

// Cериализация.

formatter.Serialize(stream, auto);

stream.Close();

ДЕСЕРИАЛИЗАЦИЯ

stream = File.OpenRead("CarData.dat");

// Десериализация.

auto = formatter.Deserialize(stream) as Mersedes;

Собственный алгоритм сериализации

1-необходимо унаследовать интерфейс ISerializable

[Serializable]

public class Car : ISerializable

после реализовать один метод

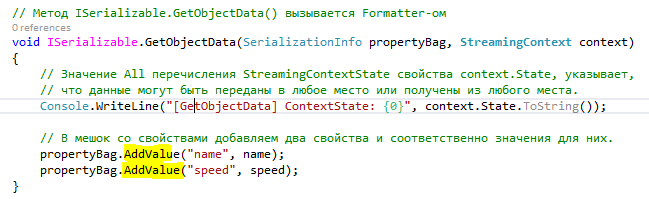
Метод ISerializable.GetObjectData() вызывается Formatter-ом

void ISerializable.GetObjectData(SerializationInfo propertyBag, **StreamingContext** context)

**StreamingContext** context - Значение All перечисления StreamingContextState свойства context.State, указывает, что данные могут быть переданы в любое место или получены из любого места

SerializationInfo propertyBag – (мешок) куда будем помещать данные которые необходимо сохранить (ключ, значение)

propertyBag.AddValue("name", name); - свойство и значение для него



2-для десериализации необходимо в классе создать закрытый пользовательский конструктор который будет по своей сигнатуре с методом GetObjectData(SerializationInfo propertyBag, **StreamingContext** context)

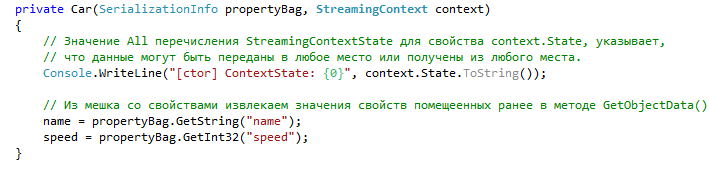
private Car(SerializationInfo propertyBag, **StreamingContext** context)

В конструкторе

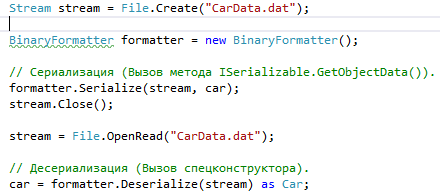
Из мешка со свойствами извлекаем значения свойств помещеенных ранее в методе GetObjectData()

name = propertyBag.GetString("name");

speed = propertyBag.GetInt32("speed");



Процесс вызова сериализации и десириализации идентичны как и при обычной сериализации

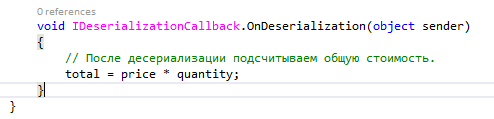


Интерфейс вызывающий метод после сериализации объекта

IDeserializationCallback

[NonSerialized]

public decimal total;



[OptionalField]

Поле добавленное в класс в новой версии. Инициализируется значением по умолчанию.

В случае если при сериализации объекта отсутствовало какое либо поле, а в дальнейшем оно было добавлено в тело класса то его необходимо пометить атрибутом [OptionalField], для того чтобы не было исключения при десириализации



}

Атрибуты для методов СЕРИАЛИЗАЦИИ

OnDeserializedAttribute

Используйте OnDeserializedAttribute, - При применении к методу указывает, что метод вызывается сразу после десериализации объекта в графе объекта.

// Например, если десериализуемый тип не содержит конструктор, создайте метод,

// чтобы задать значения любых полей в экземпляре и применить атрибут к методу.

// Чтобы использовать OnDeserializedAttribute, метод должен содержать параметр StreamingContext.

[OnDeserialized]

void CheckTotal(**StreamingContext** context)

{

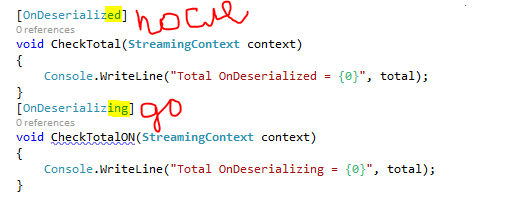
if (total == 0)

CalculateTotal(context);

else

Console.WriteLine("Total != 0");

}





[OnSerialized] – до

[OnSerializing] - после

OnSerializingAttribute

Используйте OnSerializingAttribute, чтобы выполнять действия с объектом до выполнения сериализации.

Чтобы использовать OnSerializingAttribute, метод должен содержать параметр StreamingContext.

[OnSerializing]

void CalculateTotal(**StreamingContext** context)

{

total = price \* quantity;

Console.WriteLine("Total = {0}", total);