DtoGen Documentation

Introduction

DtoGen stands for Data Transfer Objects Generator. It is a simple tool for generating C# classes that are decorated with XML and DataContract attributes for storing and representing data. The tool also generates serializers for storing and retrieving data through the generated DTOs.

There are three ways of specifying DTOs that will be generated. One of them is using XML with an XML schema, another is using a simple language that denotes DTOs in a C-like style. The last but not the least is a graphical tool similar to UML notation except that it is a little more limited than the former methods.

Using textual notation

Specifying DTOs through XML

This is the recommended way since enough editors out there have built-in support for XML and XML Schema. If given a schema the editor will automatically validate constraints, this works best because the DtoGen tool would not have to be invoked multiple times to resolve issues because they will be reported by the editor during design time.

The XML Schema can be found at http://storage.andrei15193.ro/public/dtoGenSchema.xsd. Basically the root element is called *dtoMap* with an optional attribute *namespace*. The *namespace* attribute specifies the C# *namespace* in which classes will be generated. A *namespace* attribute value must be a valid C# qualified identifier.

The root then contains any number of *dto* elements each of them having a mandatory *name* attribute. Names are IDs and must be unique within the document, they must be valid C# *identifiers*. These names in turn will become class names.

Each *dto* element can contain any number of *attribute* elements, these represent C# properties. An *attribute* has a name which has to be a valid C# *identifier* and unique within a *dto* element. Along the *name*, attributes also have a *type* attribute which can be equal to any value of a *name* attribute of a *dto* element with the exception that a *dto* may not form any sort of circular reference. In case there is need to have attributes of primitive types there are 4 predefined type: *text* (C# string), *int* (C# int), *float* (C# double) and *dateTime* (System.DateTime). Besides *name* and *type*, attributes have an optional *multiplicity* attribute which by default has the value *single* however it can be set to be a *collection*.

Single attributes hold one value of the specified type, they are get and set properties in the resulting C# code while collection attributes are get only properties of List type holding values of the specified *type* by the *attribute* element.

Specifying DTOs through DTOlang

DTOlang is a lightweight language designed only to specify dtos in a more simplistic meaner than one would do in XML. There is no root element, a DTOlang file (.dto file) has an optional namespace specification at the top and then followed by any number of dto specifications.

To specify the namespace simply write namespace followed by the namespace name which must be a valid C# qualified identifier.

To specify a dto simply write dto followed by its name which must be a valid C# identifier then by on open curly bracket (‘{‘).

Next are the attributes of the dto. To specify an attribute simply write its name which must be a valid C# identifier followed by a colon (‘:’) and then by the attribute’s type. Again this can be the name of another dto inside the same DTOlang file or one of the primitive types: text (C# string), int (C# int), float (C# double) or dateTime (System.DateTime). By default attributes have the single multiplicity. To specify a collection simply add an asterisk (‘*’) after the attribute type.

To complete the specification of a dto simply close with a curly bracket (‘}’) which will match the one opened after the dto’s name.

The same restrictions hold as for XML documents, dto names must be unique within the same file, dto attribute names must be unique within a dto and there must be no circular reference, of any sort, between dtos.

Examples

The following example specifies the same map of DTOs using both XML notation and DTOlang. The similarity between the both.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<dtoMap xmlns="http://storage.andrei15193.ro/public/dtoGenSchema.xsd"
        namespace="Examples.First">
  <dto name="Dto1">
    <attribute name="attribute1" type="Dto3" />
    <attribute name="attribute2" type="int" multiplicity="collection" />
  </dto>
  <dto name="Dto2">
    <attribute name="attribute1" type="Dto1" multiplicity="collection" />
  </dto>
  <dto name="Dto3">
    <attribute name="attribute1" type="text" />
  </dto>
</dtoMap>
```

Figure 1: DTO specification in XML.

```
namespace Examples.First

dto Dto1{
    attribute1: Dto3
    attribute2: int*
}
dto Dto2{
    attribute1: Dto1*
}
dto Dto3{
    attribute1: text
}
```

Figure 2: DTO specification in DTOlang.

Using the graphical tool

The available graphical tool uses a UML-like graphical notation to specify DTOs and relation between them. A DTO element has a name can have any number of attributes where each is of primitive type (text, int, float or dateTime) and can refer any number of other DTOs.

Each attribute has three properties, the name, its type and its multiplicity.

The graphical tool is used to generate DTOlang files that contain the corresponding textual notation to the graphical one. Below is an example of a graphical notation.

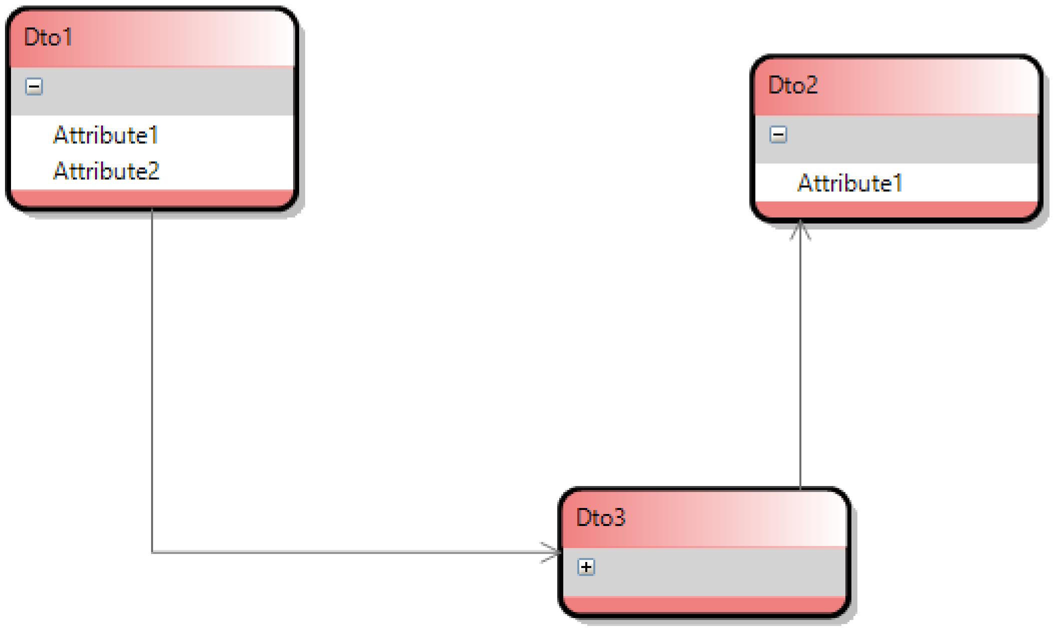

Figure 3: a graphical notation for a DTO map.

Generating C# code

To generate C# code one must use the DtoGen utility. It is a console application that takes any number of files as arguments. If the input file exists and is either a .xml or .dto file then the application will output C# source code files in the directory from where the application is run. Optionally one can set the resulting directory using the –folderPath:<path> argument (e.g.: -folderPath:..\Temp). If a namespace is specified then all resulting classes are placed within that namespace. All generated classes are partial to allow custom extension (implementing interfaces, adding members, possibly validation etc.).

The application generates for each specified DTO a C# source code file (.cs) and for each DTO root generates XML and DataContract serializers which can retrieve and store DTOs. A DTO is considered root when it is not being referred by any other DTO similarly to XML document roots, they are not contained by any other XML element.