

## EDI spec DIY guide

Last updated on 23 Jan 2017

EdiFabric uses classes as formal grammar to parse and create EDI messages. Each EDI transaction set is represented by one message class.

The purpose of each class is to convert the flat EDI structure into a hierarchical object graph by adhering to the rules described below. This means that the same EDI message can be represented in two different ways by using two separate versions of the same spec.

Having full control over the desired representation of EDI allows the consumers of the framework to amend the classes accordingly, thus supporting every format or dialect.

The following rules apply when working with any EDIFACT and X12 class:

1. Interchange headers\trailers (ISA, UNB, etc.) are hard coded and therefore static. Any amendment to them requires code changes to the framework.

2. The framework loads every class dynamically with `Type.GetType`, by finding it in the Rules project (as defined in the \*.config or passed in explicitly). Each class is identified with its namespace, which is constructed as follows:

Custom Prefix(“**EdiFabric.Rules**” by default)  
Format (X12 or Edifact)  
Version (002040 or D96A)  
Tag (810 or INVOIC)

Example: “EdiFabric.Rules.EdifactD00AINVOIC”

3. The class name holds the message type, prefixed with “**M\_**”.

Example: “public class M\_810”

4. All types and properties for each of the sub message elements (segment, group, complex element and simple element) have the following structure:

**{prefix}\_{name}\_{postfix}**

where:

**prefix** is any of {G, S, C, D} or {A, U} (HIPAA only)

**name** is free text

**postfix** is optional and is usually a number. It is used to declare types with the same name more than once.

5. Every segment is prefixed with an “S\_” and the name of its type must be the same as the name of the public property.

Example:

```
property: public S_ST S_ST  
class: public class S_ST
```

6. Every repeating segment is represented as a generic list of segments.

Example:

```
public List<S_NTE> S_NTE
```

7. Every group\loop is prefixed with “G\_”. Same property name is equal to the name of the type rule applies.

Example:

```
public G_N1 G_N1
```

Groups are used to represent ordered sequences of nodes. When representing unordered sequence, use “A\_” instead. Unordered sequences can only contain a single repetition of the node. Should multiple repetitions are required, wrap the multi-repetition in a “U\_” node.

Example:

```
public class A_DTP {  
    [XmlAttribute(Order=0)]  
    public S_DTP S_DTP {get; set;}  
    [XmlElement(Order=13)]  
    public U_DTP U_DTP {get; set;} // this can only be a single instance, no lists are  
                                   // allowed in “A_”  
}
```

```
public class U_DTP {  
    [XmlElement("S_DTP2",Order=0)]  
    public List<S_DTP2> S_DTP2 {get; set;} // this can be repeated multiple times  
}
```

8. Every repeating group is represented as a generic list of groups.

Example:

```
public List<G_N1> G_N1
```

9. Every complex element is prefixed with “C\_” and the name of its type must be the same as the public property name.

Example:

```
property: public C_C507 C_C507
```

class: public class C\_C507

10. Every repeating complex element is represented as a list of complex elements.

Example:

```
public List< C_C507> C_C507
```

11. Every simple data element is prefixed with “D\_” and can be either a string or an enum. Enums are just references to an enum declared further down in the file and can be of any name.

Example:

```
string: public string D_3148_1
```

```
enum: public EDIFACT_ID_3155 D_3155_2
```

12. Enum types are decorated with enum attribute when containing numbers.

Example:

```
public enum X12_ID_98
```

```
{
```

```
[XmlAttribute("11")]
```

```
Item11,
```

```
AC,
```

```
AB
```

```
}
```

13. Every message class can only contain groups or segments or lists of them.

14. Every group class can only contain segments or lists of them.

15. Every segment class can only contain complex or simple elements.

16. Every complex element class can only contain simple elements.

17. Every simple element class should be either string or enum.

18. All public properties in a class should be decorated with **XmlAttribute** with **Order**.

Example:

```
[XmlAttribute(Order = 1)]
```

```
public S_BGM S_BGM
```

The order drives how the public properties will be sorted after the spec is loaded.

19. Every class should be decorated with **XmlAttribute** and **XmlAttribute** with

Namespace.

Example:

```
[XmlTypeAttribute(AnonymousType = true, Namespace = "www.edifabric.com/edifact")]  
[XmlRootAttribute(Namespace = "www.edifabric.com/edifact", IsNullable = false)]  
public partial class S_UNH
```

where the namespaces are defined as:

for X12: **www.edifabric.com/x12**

for Edifact: **www.edifabric.com/edifact**

20. Follow the same rules when creating an XSD, with the following pointers:

- XSD "sequence" is represented with groups "G\_"
- XSD "all" is represented with "A\_"
- Repetitions in "all" are wrapped in "U\_"
- XSD must be valid