This is a description of a serious problem (Issue 1) in the PL/I and Cobol code generated by the DFHWS2LS program. Further two other issues is described.

## ISSUE 1

The following example shows that the numbering mechanism used for the generated data structures has some very serious consequences.

**PROBLEM:**

Variable names in the generated PL/I / COBOL code is unnecessary added a number where it is not necessary for unique identification of the fields.

**CONSEQUENCES:**

1) It is not possible to reuse the program code that is written to map from the generated PL/I or Cobol code to a corresponding PL/I or Cobol Com Area.
2) When  small changes in the XML Schema are introduced (like inserting an element) then
   - Need for refactoring/rewriting greater parts of the mapping code can occur
   - Great risk of introducing errors in the mapping code can occur. Furthermore these errors are very hard to find.

**REQUESTED SOLUTION:**

- The **REUSE** of mapping code that maps between the same XML Scema definition (CustomerInfoType in the wsdl) and a corresponding PL/I (COBOL) com area structure in the two operations should be possible. **This is a must**. Otherwise the same mapping must be coded again and again with the obvious problems redundant code creates, both keeping it in sync and the productivity penalty this creates.....!!!
- The addition of numbers in the above shown situations is unnecessary as the field names will be unique by using the upper levels to identity the field you want to use. So these numbers should not be generated!!!

**PROBLEM IN DETAIL:**

**Description of the case**

The following webservice defintion (Example1_InputOutput.wsdl) with the operation 'GetCustomerInfo' has the response 'CustomerInfo'. The CustomerInfoType is defined like this:

```
	<complexType name="CustomerInfoType">
	<annotation><documentation>
		</documentation></annotation>
		<sequence>
			<element name="CustomerId" type="integer"></element>
			<element name="Name" type="string" minOccurs="1" maxOccurs="1"></element>
			<element name="Street" type="string" minOccurs="1" maxOccurs="1"></element>
			<element name="HouseNo" type="string" minOccurs="1" maxOccurs="1"></element>
			<element name="ZipCodeCity" type="tns:CodeValueType" minOccurs="1" maxOccurs="1"></element>
			<element name="Country" type="tns:CodeValueType" minOccurs="1" maxOccurs="1"></element>
			<element name="Status" type="tns:CodeValueType" minOccurs="1" maxOccurs="1"></element>
			<element name="SaleToCustomerYTD" type="tns:SaleToCustomerType" minOccurs="0" maxOccurs="1"></element>
			<element name="SaleToCustomerLastYear" type="tns:SaleToCustomerType" minOccurs="0" maxOccurs="1">
</element>
		</sequence>
	</complexType>

	<complexType name="CodeValueType">
	<annotation><documentation>
			Code/Value used for coded fields, where the human readable text is communicated as well. The OutOfUse
indicates whether
			the Code is usable for new instances, or is only used for retrieval of existing instances.
	</documentation></annotation>
		<sequence>
			<element name="Code" type="string"></element>
			<element name="Value" type="string" minOccurs="0" maxOccurs="1"></element>
			<element name="OutOfUse" type="boolean" minOccurs="0" maxOccurs="1"></element>
		</sequence>
	</complexType>

	<complexType name="SaleToCustomerType">
	<annotation><documentation>
			Specifies for a year how much we have sold to the customer
	</documentation></annotation>
		<sequence>
			<element name="Turnover" type="decimal" minOccurs="1" maxOccurs="1"></element>
			<element name="NumberOfSales" type="integer" minOccurs="0" maxOccurs="1"></element>
		</sequence>
	</complexType>
```

However after the development has finished a new demand from the business means that two new elements must be added to the CustomerInfoType so now (Example2 _InputOutput.wsdl) has the following definition of CustomerInfoType:

```
      <complexType name="CustomerInfoType">
      <annotation><documentation>
            </documentation></annotation>
            <sequence>
                  <element name="CustomerId" type="integer"></element>
                  <element name="CustomerCategory" type="tns:CodeValueType" minOccurs="1" maxOccurs="1"></element>
                  <element name="Name" type="string" minOccurs="1" maxOccurs="1"></element>
                  <element name="Street" type="string" minOccurs="1" maxOccurs="1"></element>
                  <element name="HouseNo" type="string" minOccurs="1" maxOccurs="1"></element>
                  <element name="ZipCodeCity" type="tns:CodeValueType" minOccurs="1" maxOccurs="1"></element>
                  <element name="Country" type="tns:CodeValueType" minOccurs="1" maxOccurs="1"></element>
                  <element name="Status" type="tns:CodeValueType" minOccurs="1" maxOccurs="1"></element>
                  <element name="SaleToCustomerBudget" type="tns:SaleToCustomerType" minOccurs="0" maxOccurs="1"></element>
                  <element name="SaleToCustomerYTD" type="tns:SaleToCustomerType" minOccurs="0" maxOccurs="1"></element>
                  <element name="SaleToCustomerLastYear" type="tns:SaleToCustomerType" minOccurs="0" maxOccurs="1">
</element>
            </sequence>
      </complexType>

      <complexType name="CodeValueType">
      <annotation><documentation>
                  Code/Value used for coded fields, where the human readable text is communicated as well. The OutOfUse
indicates whether
                  the Code is usable for new instances, or is only used for retrieval of existing instances.
      </documentation></annotation>
            <sequence>
                  <element name="Code" type="string"></element>
                  <element name="Value" type="string" minOccurs="0" maxOccurs="1"></element>
                  <element name="OutOfUse" type="boolean" minOccurs="0" maxOccurs="1"></element>
            </sequence>
      </complexType>

      <complexType name="SaleToCustomerType">
      <annotation><documentation>
                  Specifies for a year how much we have sold to the customer
      </documentation></annotation>
            <sequence>
                  <element name="Turnover" type="decimal" minOccurs="1" maxOccurs="1"></element>
                  <element name="NumberOfSales" type="integer" minOccurs="0" maxOccurs="1"></element>
            </sequence>
      </complexType>
```

The only difference is that the element `CustomerCategory`has been added just after CustomerId and the element `SaleToCustomerBudget`has been added just after the Status element.

As the `CustomerCategory`is a simple code-value structure the type-definition `CodeValueType`is used for this elememt.

However now the generated code changes so that the generated code (e.g. PL/I) for the *Status*element changes from (Example1):

```
09 Status,
  12 Code                             CHAR(255) VARYING
UNALIGNED,

  12 Value2_num                       UNSIGNED FIXED BINARY(32)
UNALIGNED,

  12 Value                            CHAR(255) VARYING
UNALIGNED,

  12 OutOfUse2_num                    UNSIGNED FIXED BINARY(32)
UNALIGNED,

  12 OutOfUse2,
    15 *                                BIT (7),
    15 OutOfUse                         BIT (1),
```

to (Example2):

```
09 Status,
  12 Code                             CHAR(255) VARYING
UNALIGNED,

  12 Value3_num                       UNSIGNED FIXED BINARY(32)
UNALIGNED,

  12 Value                            CHAR(255) VARYING
UNALIGNED,

  12 OutOfUse3_num                    UNSIGNED FIXED BINARY(32)
UNALIGNED,

  12 OutOfUse3,
    15 *                                BIT (7),
    15 OutOfUse                         BIT (1),
```

It is now obvious that the mapping code that maps the data in the

```
09 Status,
  12 Code                             CHAR(255) VARYING
```

field in the generated data structure to a corresponding field in another PL/I data structure that is used for further processing must be changed.This is necessary because the information about the multiplicity for this field is no longer in the Value2_num but in Value3_num field.

**The consequences are:**

- The mapping of all Value fields (defined in the CodeValueType defintion in the XML Schema) must now be corrected. Add 1 to the 'X' in the 'ValueX_num' variable name...... This is a hopeless task as the introduction of errors is an almost certain thing. And WORSE these errors is very very hard to find as the compiler will not help you and you have to construct testdata where the Value fields one by one should be given the multiplicity 0 in order to be sure that you have changed all the mappings correctly.

- Another obvious consequence is that you cannot reuse your mapping code. If you have another operation on the same (or another) webservice that have a response consisting of two elements e.g. PersonInfo and CustomerInfo (specified in that order) then the mapping code coded for the GetCustomerInfo operation is unusable if PersonInfo have one or more elements with the type *CodeValueType*(or *SaleToCustomerType*). It cannot be reused as the numbering of the ValueX_num fields is different in the two generated code versions of the CustomerInfo structure.

**Requested Solution:**

- The REUSE of mapping code that maps between the same XML Scema definition (CustomerInfoType in the wsdl) and a corresponding PL/I (COBOL) com area structure in the two operations is a must. Otherwise the same mapping must be coded again and again with the obvious problems redundant code creates, both keeping it in sync and the productivity penalty this creates.....!!!

- The addition of numbers in the above shown situations is unnecessary as the field names will be unique by using the upper levels to identity the field you want to use. So these numbers should not be generated!

## ISSUE 2

**Numbering principle differs between generated PL/I and COBOL code:**
NOTE: If the addition of numbers is removed as suggested in Issue 1 then this issue is not relevant. However we mention this observation to be sure that you are aware of this.

There is a difference between the generated PL/I and COBOL code concerning the addition of numbers to variable names. In the example above the "`12 Value CHAR(255) VARYING`" line from the generated PL/I has no number added to the 'Value' variable name (which is not necessary by the way). It is only the multiplicity variable 'Value2-num' that has the number added (as I see it this should NOT be necessary, the 'Value_num' should be all that is necessary as described in Issue 1).

However in the generated Cobol both the 'Value' variable and the Value-counter variable ('Value2-num') has the number added, like this:

```
       09 XStatus.
         12 XCode2-length                 PIC S9999 COMP-5
    SYNC.
         12 XCode2                        PIC X(255).

         12 Value2-num                    PIC S9(9) COMP-5
    SYNC.

         12 Value2.
           15 XValue3-length                PIC S9999 COMP-5
    SYNC.
           15 XValue3                       PIC X(255).

         12 OutOfUse2-num                 PIC S9(9) COMP-5
    SYNC.

         12 OutOfUse2                     PIC X DISPLAY.
```

(the X in front of 'Status' and 'Value' is added (correctly) to avoid using the corresponding reserved words in Cobol).

Here it should be noted that the numbering gets very confusing as the numbers gets out-of-sync (theres is '`12 Value2.`' but the enclosed fields '`15 XValue3-length`' and '`15 XValue3`'). This is also an error generator when humans have to produce mapping code to/from such a structure.

It is the same for the DECIMAL data type (and not just for the 'string' data type showed above).
From the example 1 the '`<element name="Turnover" type="decimal" minOccurs="1" maxOccurs="1"></element>`' is generated to PL/I like this

```
09 SaleToCustomerLastYear,
            12 Turnover                       FIXED DECIMAL (31,3),
```

and in COBOL to:

```
               09 SaleToCustomerLastYear.
                 12 Turnover1                     PIC S9(15)V9(3)
            COMP-3.
```

Why the difference? Why addition of the number as it is unnecessary?

## ISSUE 3:

**When you use the XML Schema datatype 'DECIMAL' this is generated differently to PL/I and COBOL**

As can be seen in the last example in Issue 2 the 'DECIMAL' datatype is generated to 'FIXED DECIMAL  (31,3)' in PL/I and to 'PIC S9(15)V9(3)'. Why this difference ?

**DOCUMENTATION:**

**Enclosed files:**

Example wsdl with a data structure definition (CustomerInfoType)
Example1_InputOutput.wsdl

Same as Example1 but with two elements inserted
Example2_InputOutput.wsdl

Example1_InputOutput.wsdl          Example2_InputOutput.wsdl

Generated response language structure(s) for WSDL operation 'GetCustomerInfo':

Example1:

| | |
|---|---|
| EXAM1001-PLI.txt | (PL/I) |
| EXAM1001-CBL.txt | (Cobol) |

Example2:

| | |
|---|---|
| EXAM2001-PLI.txt | (PL/I) |
| EXAM2001-CBL.txt | (Cobol) |

EXAM2001-PLI.txt EXAM1001-CBL.txt EXAM1001-PLI.txt EXAM2001-CBL.txt

-------------------- o --------------------