We Translate
Business Processes

from the Mind
to the Computer
to the Bottom Line.

BUSINESS & COMPUTERS, Inc.
13839 Mur-Len Rd, Suite M
OLATHE, KANSAS 66062

Phone: (913) 764-2311
Fax: 764 7515
larryg@kcnet.com

What is an XML Document and How Can It be Used in a Stored Procedure

A note – the below is my humble opinion – with testing – If you use my ideas please test them and if you have problems or learn more let me know.

What is an XML Document? (XML was five years old on Feb-10-2003)

An XML document (aka Extensible Markup Language) is a technique for creating structured data in a text file. It is the de facto technology used in transferring data between two different systems or programs. Even if the other system uses Lenox and you use Windows with SQL Server, you can communicate with each other. If you haven't been asked to furnish your data in XML format, or import XML data into your system, you have a job where you don’t communicate with the world outside your system. If you see yourself eventually communicating outside your system, you will eventually need to learn XML.

In the old days, you and I would agree on a format for a text file and we would pass the data back and forth using this text file. In a text file we might agree to the following record:
First_Name = space 1 to 20 Last_Name = 21 to 40 Address = 41 to 60 City = 61 to 75 State =76 to 77 etc

```
1234567890123456789012345678901234567890123456789012345678901234567890
Larry               Gordon              13839 MurLen Suite MOlathe         KS
```

First of all, especially with a big file, I have to explain what fields go where, and other rules I implement. Once we agree on the format, we might communicate using this file for 5 years. You don’t know it, but I had a new partner come to me and say they had to have a middle initial in the file. Without thinking I say “No Problem.” I then make a minor change to my program so that I put the middle initial in space 21. From that point on the file doesn’t work for you.

With XML, that’s not a problem. We show field names and if I add fields, your program still works. It’s almost like importing an Excel Spreadsheet even though the data comes from two non related systems.

With XML, that's not a problem. We show field names and if I add fields, your program still works. It's almost like importing an Excel Spreadsheet even though the data comes from two non related systems.

Element Based XML without middle initial

```
<?xml version="1.0"?>
<CustomerName>
        <Details>
            <First_Name>Larry</First_Name>
            <Last_Name>Gordon</Last_Name>
            <Address>13839 Mur-Len  Suite M</Address>
            <City>Olathe</City>
            <State>KS</State>
        </Details>
        <Details>
            <First_Name>Joe</First_Name>
            <Last_Name>Blow</Last_Name>
            <Address>123 Main St</Address>
            <City>Kansas City</City>
            <State>MO</State>
        </Details>
</CustomerName>
```

Element Based XML with middle initial

```
<?xml version="1.0"?>
<CustomerName>
        <Details>
            <First_Name>Larry</First_Name>
            <MiddleInitial>J</MiddleInitial>
            <Last_Name>Gordon</Last_Name>
            <Address>13839 Mur-Len  Suite M</Address>
            <City>Olathe</City>
            <State>KS</State>
        </Details>
        <Details>
            <First_Name>Joe</First_Name>
            <MiddleInitial>P</MiddleInitial>
            <Last_Name>Blow</Last_Name>
            <Address>123 Main St</Address>
            <City>Kansas City</City>
            <State>MO</State>
        </Details>
</CustomerName>
```

XML is like dealing in a society where everyone speaks the same language. There are different dialects of XML, but it is still easy to deal with XML, even with the different dialects. The two main dialects are "Attribute-centric" XML and "Element-centric" XML. Above is Element-Centric or Element Base XML. On the next page you will see Attribute Centric or Attribute based XML

Attribute Based XML

```
<?xml version="1.0"?>
<rs:data>
            <z:row MonthNumb='1' MonthAbbrev='JAN' Month='January'/>
            <z:row MonthNumb='2' MonthAbbrev='FEB' Month='February'/>
            <z:row MonthNumb='3' MonthAbbrev='MAR' Month='March'/>
            <z:row MonthNumb='4' MonthAbbrev='APR' Month='April'/>
            <z:row MonthNumb='5' MonthAbbrev='MAY' Month='May'/>
            <z:row MonthNumb='6' MonthAbbrev='JUN' Month='June'/>
            <z:row MonthNumb='7' MonthAbbrev='JUL' Month='July'/>
            <z:row MonthNumb='8' MonthAbbrev='AUG' Month='August'/>
            <z:row MonthNumb='9' MonthAbbrev='SEP' Month='September'/>
            <z:row MonthNumb='10' MonthAbbrev='OCT' Month='October'/>
            <z:row MonthNumb='11' MonthAbbrev='NOV' Month='November'/>
            <z:row MonthNumb='12' MonthAbbrev='DEC' Month='December'/>
</rs:data>
```

You will see the Attribute Based XML in earlier versions of Microsoft products, such as

* Microsoft Office 2000	*Internet Explorer 5.0
* SQL Server 2000	* MS BizTalk Framework

Also in the same earlier versions of Microsoft's products, the Schema that is used is called **XDR** or XML Data Reduced. This was a format that Microsoft came up with, that did not comply with the W3C's XML Schema. (W3C is the ruling organization for all XML http://www.w3.org/XML/)

The newest XML Schema is **<u>XSD</u>** or 'XML Schema Definition' "which offers facilities for describing the structure and constraining the contents of XML 1.0 documents". The W3C XML Schema specification has advanced to the 'Proposed Recommendation' for XML 1.0 documents. Microsoft has committed to this format, and is using it in .Net and most of the future Microsoft products. XSD does a lot to define the data XML is sending. In fact we are about to get to the place, (we are not there yet.) where you could import an XML document into your database and it would build a table with the right data types and constraints.

XML with XSD Schema Definition

```
<?xml version="1.0" encoding="utf-8"?>
<DataSet xmlns="http://tempuri.org/">
  <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="Orders">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="OrderID" type="xs:int" minOccurs="0" />
                <xs:element name="OrderDate" type="xs:dateTime" minOccurs="0" />
                <xs:element name="RequiredDate" type="xs:dateTime" minOccurs="0" />
                <xs:element name="ShippedDate" type="xs:dateTime" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
    <NewDataSet xmlns="">
      <Orders diffgr:id="Orders1" msdata:rowOrder="0">
        <OrderID>10643</OrderID>
        <OrderDate>1997-08-25T00:00:00.0000000-05:00</OrderDate>
        <RequiredDate>1997-09-22T00:00:00.0000000-05:00</RequiredDate>
        <ShippedDate>1997-09-02T00:00:00.0000000-05:00</ShippedDate>
      </Orders>
      <Orders diffgr:id="Orders2" msdata:rowOrder="1">
        <OrderID>10692</OrderID>
        <OrderDate>1997-10-03T00:00:00.0000000-05:00</OrderDate>
        <RequiredDate>1997-10-31T00:00:00.0000000-06:00</RequiredDate>
        <ShippedDate>1997-10-13T00:00:00.0000000-05:00</ShippedDate>
      </Orders>
    </NewDataSet>
  </diffgr:diffgram>
</DataSet>
```

For more information about the schemas you might want to look at http://www.oasis-open.org/cover/schemas.htm

A very simple book that is great for those starting to work with XML is "XML the Microsoft Way" by Peter G. Aitken.

We could go on for many more pages, but this will give you a primer on XML.

What kind of project caused me to use Open XML in Stored Procedures

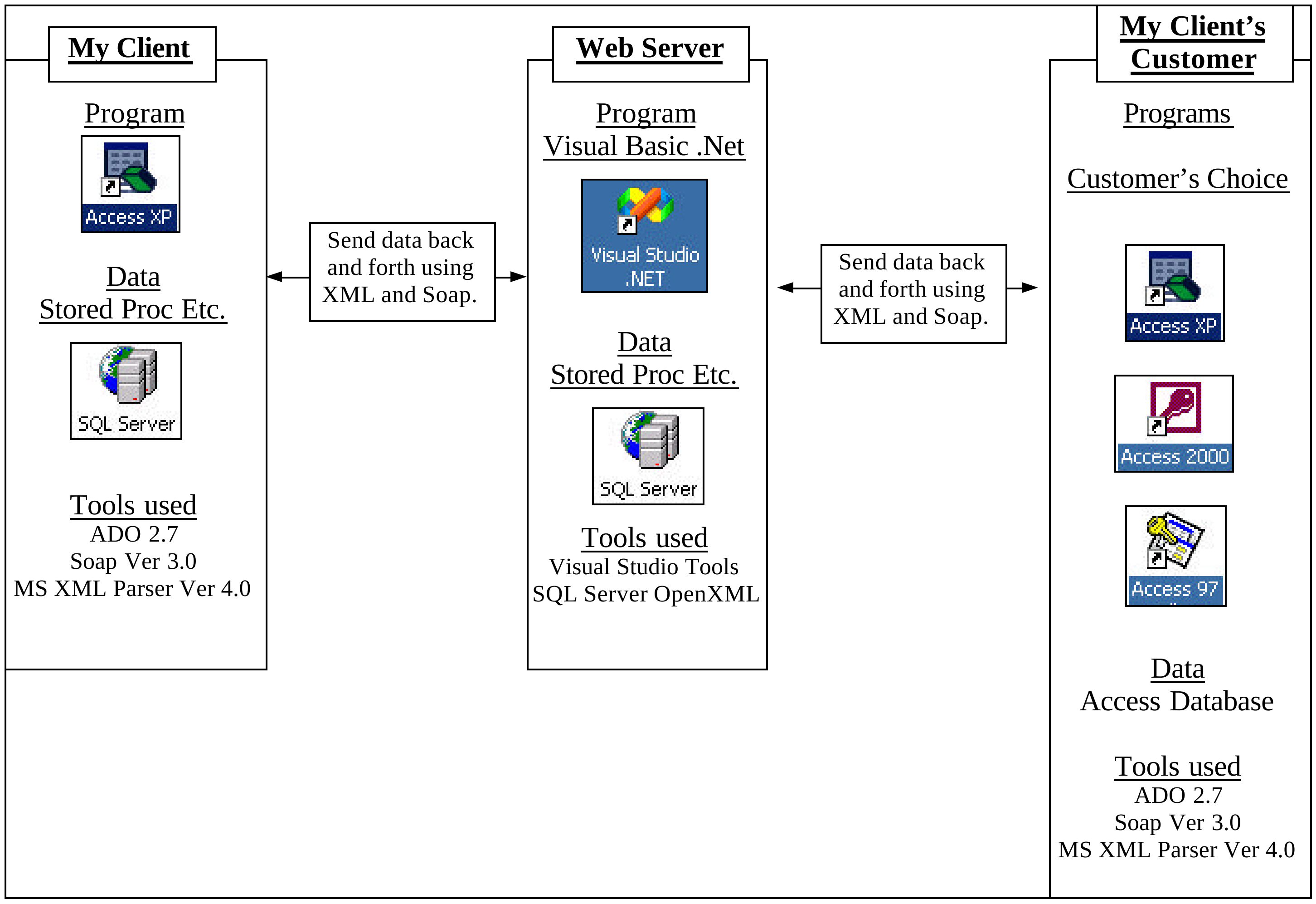
I have been using XML inside my applications for over 3 years. The first time was getting a trucking company name and contact information by passing a zip code to a website. My experience with the web and users was that the internet was too slow to do this process in real time. I was pleasantly surprised.

I thought the user would be waiting long enough to get a cup of coffee from the time they pushed the button, and the data was sitting in their form. I was wrong. I was use to bringing back graphics from the web. When you bring back data, the process is quick. What happened is that the first time a person pushed the button it did take 15 to 20 seconds (*this is too long*), but each time they pushed the button after that (with different zip codes each time) it was 3 to 5 seconds, which I saw as acceptable to have up to the minute information.

I used XML many times since my first application, but my last project was the first one using OpenXML in SQL Server. The diagrams below describe how the process works.

Basically if your company is like most companies, they like to keep track of information that is communicated back and forth between their partners. With the below system for example, we are dealing with Returned Goods from many manufactures. We track the goods being returned, the people who requested the returns, the pick up location, the delivery location, all the transportation information, and complete documentation of the communication between all parties.

With the system in place both parties, my client and my client's customers, have complete information about each transaction inside the database sitting on their computers. My client gives the customer the database, and the customer sends and receives the data through the web. This makes my client more competitive in the marketplace.



Open XML (using XML in a Stored Procedure)

Now that we know what XML is, how the heck do we transfer the data into our tables inside SQL Server? I looked at a number of technologies and decided to use OpenXML with SQL Server. We could bring multiple records into SQL Server at the same time. The old way to do this was to parse a record, and put one record at a time into a SQL Server table. With OpenXML we can put multiple records into a table at one time.

As you look at OpenXML, SQL Server gives a number of choices that basically either look at Attribut-Centric XML or Element-Centric XML. I will be looking at Element-Centric XML, however it would take very little to change the following code to Attribute based. (Our Flag will be #2, Element based)

SQL Chart #1

Byte Value	Description
0	Defaults to attribute-centric mapping.
1	Use the attribute-centric mapping. Can be combined with XML_ELEMENTS; in which case, attribute-centric mapping is applied first, and then element-centric mapping is applied for all columns not yet dealt with.
2	Use the element-centric mapping. Can be combined with XML_ATTRIBUTES; in which case, element-centric mapping is applied first, and then attribute-centric mapping is applied for all columns not yet dealt with.
8	Can be combined (logical OR) with XML_ATTRIBUTES or XML_ELEMENTS. In context of retrieval, this flag indicates that the consumed data should not be copied to the overflow property @mp:xmltext.

Simple Transact-SQL using OpenXML

Hug_OpenXML_01_XMLStart.sql

@intXMLdocNbr is the document handle of the internal representation of an XML document.

@vcXMLdoc is a variable to store the XML document. In a stored Proc, we would use text.

declare @vcXMLdoc text

We set @vcXMLdoc = to our XML document.

```
declare @intXMLdocNbr int
declare @vcXMLdoc varchar(1000)
set @doc ='
<?xml version="1.0"?>
<CustomerName>
  <Details>
    <Customer_ID>255</Customer_ID>
    <First_Name>Larry</First_Name>
    <MiddleInitial>J</MiddleInitial>
    <Last_Name>Gordon</Last_Name>
    <Address>13839 Mur-Len  Suite M</Address>
    <City>Olathe</City>
    <State>KS</State>
  </Details>
  <Details>
    <Customer_ID>256</Customer_ID>
    <First_Name>Joe</First_Name>
    <MiddleInitial>P</MiddleInitial>
    <Last_Name>Blow</Last_Na me>
    <Address>123 Main St</Address>
    <City>Kansas City</City>
    <State>MO</State>
  </Details>
</CustomerName>'
```

We create an internal representation of the XML document, inside SQL Server memory.

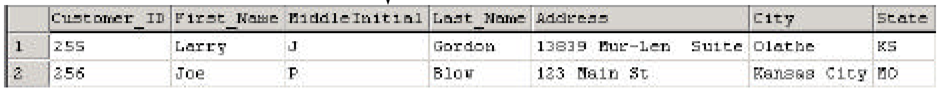
```
--Create an internal representation of the XML document.
exec sp_xml_preparedocument @intXMLdocNbr OUTPUT, @vcXMLdoc
```

Display the recordset.

```
-- SELECT stmt using OPENXML rowset provider
SELECT *
FROM   OPENXML (@intXMLdocNbr , '/CustomerName/Details',2)
        WITH (Customer_ID  int ,
                    First_Name   varchar(20)   ,
                    MiddleInitial varchar(1) ,
                    Last_Name     varchar(20)        ,
                    Address   varchar(20),
                    City      varchar(15),
                    State     Varchar(2)      )
```

We remove the XML from SQL Server memory.

```
Exec sp_xml_removedocument @idoc
```



	Customer_ID	First_Name	MiddleInitial	Last_Name	Address	City	State
1	255	Larry	J	Gordon	13839 Mur-Len Suite	Olathe	KS
2	256	Joe	P	Blow	123 Main St	Kansas City	MO

The Above Results in the following table.

FROM   OPENXML (@intXMLdocNbr,   '/CustomerName/Details',   2)

Document Handle **Row Pattern** **Flag**

Document Handle = Is the handle of the internal representation of an XML document in memory.

Row Pattern = Is the XPath pattern used to identify the nodes in the XML document whose handle is passed in the *Document Handle* parameter to be processed as rows. Open XML knows what part is the Schema, so you just have to show it where the data starts. In the above example the data starts on the Details node, so we lead OpenXML to the Details node by '/CustomerName/Details' (Note: '//Details' would work too.)

Flag = See SQL Chart #1 (previous page)

In *SchemaDeclaration* (in the WITH clause), the specified *ColName* values match the corresponding XML element names. We need to define the data type in this section. In my experience I have found that dates some time have problems moving from an XML document to SQL Server. To solve the problem, move the date to a varchar and then convert it to a date.

```
-- SELECT stmt using OPENXML rowset provider
SELECT *
FROM   OPENXML (@intXMLdocNbr , '/CustomerName/Details',2)
       WITH (Customer_ID  int ,
                First_Name   varchar(20)   ,
                MiddleInitial varchar(1) ,
                Last_Name    varchar(20)       ,
                Address   varchar(20),
                City      varchar(15),
                State     Varchar(2)     )
```

You do not need to bring in all the fields from the XML document, and you do not need to keep the same order.

```
-- SELECT stmt using OPENXML rowset provider
SELECT *
FROM   OPENXML (@intXMLdocNbr , '/CustomerName/Details',2)
       WITH (State Varchar(2),
               Customer_ID  int   )
```

We would use the below to open the XML seen in "XML with XSD Schema Definition" earlier in this document

Hug_OpenXML_02_XMLStart.sql

```
-- SELECT stmt using OPENXML rowset provider
SELECT *
FROM   OPENXML (@intXMLdocNbr, '//Orders', 2)
        WITH (OrderID   int   ,
             OrderDate varchar(20),
             RequiredDate  varchar(20) ,
             ShippedDate  varchar(20) )
```

If your data types and field names, line up with a table, you can define the schema using a table name.

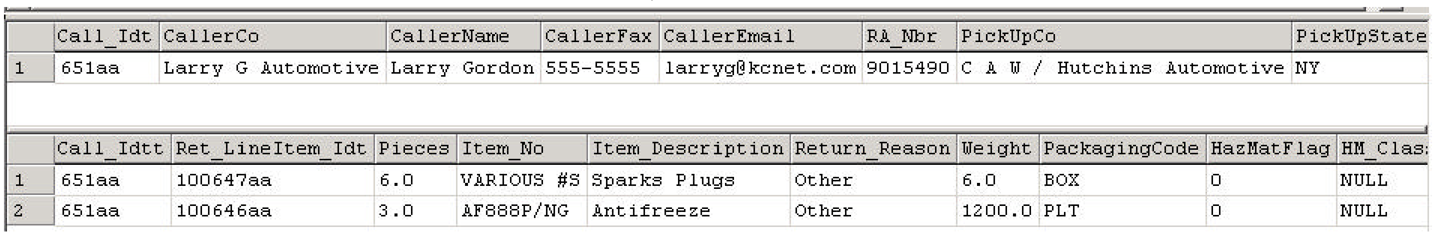
```
-- SELECT stmt using OPENXML rowset provider
SELECT *
FROM   OPENXML (@intXMLdocNbr , '/CustomerName/Details',2)
       WITH  tbl_Customers
```

XML - Shaped Recordset

Hug_OpenXML_03_Shaped.sql

```
declare @XMLDoc varchar(5000)
Declare @intXMLDocNbr int

set @XMLDoc ='
<?xml version="1.0"?>
<Returns>
  <Return>
    <Call_Idt>651aa</Call_Idt>
    <YFS_Cust_Idtt>78</YFS_Cust_Idtt>
    <CallerCo>Larry G Automotive</CallerCo>
    <CallerName>Larry Gordon</CallerName>
    <CallerFax>555-5555</CallerFax>
    <CallerEmail>larryg@kcnet.com</CallerEmail>
    <RA_Nbr>9015490</RA_Nbr>
    <PickUpCo>C A W / Hutchins Automotive</PickUpCo>
    <PickUpState>NY</PickUpState>
    <PickUpCountry>USA</PickUpCountry>
    <PickUpZip>14304</PickUpZip>
      <LineItems>
        <LineItem>
          <Call_Idtt>651aa</Call_Idtt>
          <Ret_LineItem_Idt>100647aa</Ret_LineItem_Idt>
          <Pieces>6</Pieces>
          <Item_No>VARIOUS #S</Item_No>
          <Item_Description>Sparks Plugs</Item_Description>
          <Return_Reason>Other</Return_Reason>
          <Weight>6</Weight>
          <PackagingCode>BOX</PackagingCode>
          <HazMatFlag>0</HazMatFlag>
        </LineItem>
        <LineItem>
          <Call_Idtt>651aa</Call_Idtt>
          <Ret_LineItem_Idt>100646aa</Ret_LineItem_Idt>
          <Pieces>3</Pieces>
          <Item_No>AF888P/NG</Item_No>
          <Item_Description>Antifreeze</Item_Description>
          <Return_Reason>Other</Return_Reason>
          <Weight>1200</Weight>
          <PackagingCode>PLT</PackagingCode>
          <HazMatFlag>0</HazMatFlag>
        </LineItem>
      </LineItems>
  </Return>
</Returns>'

--Create an internal representation of the XML document.
Exec sp_xml_preparedocument @intXMLDocNbr OUTPUT, @XMLDoc

-- Display the main Record
                SELECT * From   OPENXML (@intXMLDocNbr, '//Return', 2)
                               WITH (Call_Idt varchar(9),   CallerCo varchar(30),   CallerName varchar(30),  CallerFax varchar(14),   CallerEmail varchar(100),
                                        RA_Nbr varchar(30),   PickUpCo varchar(30),    PickUpState varchar(2),   PickUpCountry varchar(10),   PickUpZip varchar(15))

-- Display the Child Record
                SELECT * From  OPENXML (@intXMLDocNbr, '/Returns/Return/LineItems/LineItem',2)
                              WITH([Call_Idtt] varchar(10) ,  [Ret_LineItem_Idt] varchar(10)  ,[Pieces] float,  Item_No] varchar(40), [Item_Description] varchar(250)  ,
                              [Return_Reason] varchar(50),  Weight] float,  [PackagingCode] varchar(3)  ,[HazMatFlag] bit ,[HM_Class] varchar(50)  ,
                              [HM_Un_or_Na_Id] varchar(50) , [HM_Packing_Group] varchar(50)  )

--Remove XML document from memory.
EXEC sp_xml_removedocument @intXMLDocNbr
```

The above returns the below



```
   Call_Idt CallerCo           CallerName   CallerFax CallerEmail      RA_Nbr  PickUpCo                    PickUpState
1  651aa    Larry G Automotive Larry Gordon 555-5555  larryg@kcnet.com 9015490 C A W / Hutchins Automotive NY

   Call_Idtt Ret_LineItem_Idt Pieces Item_No    Item_Description Return_Reason Weight PackagingCode HazMatFlag HM_Clas
1  651aa     100647aa         6.0    VARIOUS #S Sparks Plugs     Other         6.0    BOX           0          NULL
2  651aa     100646aa         3.0    AF888P/NG  Antifreeze       Other         1200.0 PLT           0          NULL
```

XML - Create table in SQL Server

Hug_OpenXML_04_CreateTable.sql

```
use zHug_OpenXML
declare @intXMLdocNbr int,
          @vcXMLdoc varchar(1000)

set @vcXMLdoc ='
<?xml version="1.0"?>
<CustomerName>
  <Details>
    <Customer_ID>255</Customer_ID>
    <First_Name>Larry</First_Name>
    <MiddleInitial>J</MiddleInitial>
    <Last_Name>Gordon</Last_Name>
    <Address>13839 Mur-Len  Suite M</Address>
    <City>Olathe</City>
    <State>KS</State>
  </Details >
  <Details>
    <Customer_ID>256</Customer_ID>
    <First_Name>Joe</First_Name>
    <MiddleInitial>P</MiddleInitial>
    <Last_Name>Blow</Last_Name>
    <Address>123 Main St</Address>
    <City>Kansas City</City>
    <State>MO</State>
  </Details>
</CustomerName>'

-- Delete table if it exists
if exists (select * from sysobjects where id=object_id('[dbo].[tbl_Company]') and OBJECTPROPERTY(id, 'IsTable')=1)
   drop table [tbl_Company]

-- Create tbl_Company
Create Table tbl_Company(Customer_ID  int ,
                         First_Name   varchar(20)   ,
                           MiddleInitial varchar(1) ,
                         Last_Name     varchar(20)       ,
                           Address   varchar(20),
                           City      varchar(15),
                           State     Varchar(2))


--Create an internal representation of the XML document.
exec sp_xml_preparedocument @intXMLdocNbr OUTPUT, @vcXMLdoc

-- Put XML data into tbl_Company
INSERT INTO  tbl_Company (Customer_ID, First_Name, MiddleInitial, Last_Name, Address, City, State)
        SELECT *
                FROM   OPENXML (@intXMLdocNbr, '/CustomerName/Details',2)
     WITH (Customer_ID  int ,
        First_Name   varchar(20)    ,
        MiddleInitial varchar(1) ,
        Last_Name     varchar(20)       ,
        Address   varchar(20),
        City      varchar(15),
        State     Varchar(2))

select * from tbl_Company
--Remove XML document from memory.
Exec sp_xml_removedocument @intXMLdocNbr
```

XML - Insert into table in SQL Server

Hug_OpenXML_05_Insert_Into_Table.sql

```
use zHug_OpenXML

declare @intXMLdocNbr int,
                @vcXMLdoc varchar(1000)

set @vcXMLdoc ='
<?xml version="1.0"?>
<CustomerName>
  <Details>
    <Customer_ID>257</Customer_ID>
    <First_Name>Pete</First_Name>
    <MiddleInitial>A</MiddleInitial>
    <Last_Name>Wilson</Last_Name>
    <Address>456 Oak St.</Address>
    <City>Overland Park</City>
    <State>KS</State>
  </Details>
  <Details>
    <Customer_ID>258</Customer_ID>
    <First_Name>Sue</First_Name>
    <MiddleInitial>A</MiddleInitial>
    <Last_Name>Adams</Last_Name>
    <Address>789 151 St</Address>
    <City>Kansas City</City>
    <State>KS</State>
  </Details>
</CustomerName>'

--Create an internal representation of the XML document.
exec sp_xml_preparedocument @intXMLdocNbr OUTPUT, @vcXMLdoc

-- Put XML data into tbl_Company
INSERT INTO  tbl_Company (Customer_ID, First_Name, MiddleInitial, Last_Name, Address, City, State)
             SELECT *
                          FROM   OPENXML (@intXMLdocNbr, '/CustomerName/Details',2)
        WITH (Customer_ID  int ,
             First_Name   varchar(20)   ,
             MiddleInitial varchar(1) ,
             Last_Name     varchar(20)        ,
             Address   varchar(20),
             City      varchar(15),
              State     Varchar(2))

select * from tbl_Company


--Remove XML document from memory.
Exec sp_xml_removedocument @intXMLdocNbr
```