CHAPTER 1

INTRODUCTION TO XML SCHEMA

This chapter provides a basic introduction to XML Schema. It briefly explains what an XML schema is and its significance in today’s programming world. Further, it gives a brief overview of DTD and XDR, the predecessors of XSD. Finally, this chapter looks into the XML capabilities of SQL Server 2000, 2005 and 2008.

What is an XML Schema?

An XML Schema is a document which describes another XML document. XML Schemas are used to validate XML documents. An XML schema itself is an XML document which contains the rules to be validated against a given XML instance document.

When do we need an XML schema?

When we write a piece of code (a class, a function, a stored procedure, etc.) which accepts data in XML format, we need to make sure that the data that we receive follows a certain XML structure and should contain values which are coherent. Let us look at an example.

Assume that you are writing a function/method for an application that manages employee data. Your function is expecting the employee information in the following XML structure:

```xml
<Employee>
  <Name>
    <First>Jacob</First>
    <Middle>V</Middle>
    <Last>Sebastian</Last>
  </Name>
  <!-- Deleted other information for brevity -->
</Employee>
```

Your function needs to make sure that the caller passes correct XML data. You could make use of an XML Schema to perform this validation. An XML Schema which describes and validates the above XML document is given below:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="First"/>
              <xs:element name="Middle"/>
              <xs:element name="Last"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
By validating the XML data against this schema, you could make sure that the XML document is structured exactly the way your function expects it to be.

To summarize, we need an XML schema when we need to make sure that the XML document that we need to work with is in the expected format. Further, a schema can help to make sure that the values of elements and attributes are within the accepted range (age should be between 18 and 65, Order Date cannot be a future date, etc.) and in the required format (Phone Number should be in the format of (999) 999-9999, Zip Code should have 5 digits, Product Code should start with an upper case letter followed by 5 digits, etc.).

**Relevance of XSD**

There has been a significant increase in the popularity and usage of XML in the past few years. More and more websites and applications started adopting XML for exchanging or publishing information. A few examples are given below:

- Web sites started publishing information in the form of XML feeds (example: RSS, ATOM, RDF, etc.).
- XML Web services became an integral part of enterprise applications.
- A large number of applications are being written that make use of XML web services such as Google APIs, Amazon Web Services, etc. Many small applications that work with frequently changing information (example: news headlines, stock data, weather information, etc.) rely on XML web services.
- Most of the document formats that we use today can be converted to and from XML. Microsoft Open Office XML Format (.docx) of office 2007 and WordML of Word 2003 are examples of XML support getting into word processing. XML is extensively used for documentation. An example is the XML documentation support extended by Visual Studio.
- More and more web sites are turning to AJAX (Asynchronous Java Script and XML) programming, where data is exchanged in XML format. Many of the web pages today use XSLT to generate HTML from XML data.
- An increasing number of web sites adhere to the XHTML standard.
Many applications use xml to store session or user related data. Microsoft Dot.net applications use XML files for storing configuration data (web.config and app.config). Reporting Services stores report definitions as XML documents.

When data is managed and exchanged in XML format, there needs to be clear agreement about the structure of the XML document. Values of elements and attributes should be in the expected range as well as in the desired format. There needs to be a contract between the caller and the callee about the XML document being exchanged. Once the contract is defined, there has to be a way to enforce it and validate the XML document to make sure that it adheres to the format defined in the contract.

This is where we need an XML Schema! A Schema provides such a contract. It defines the structure of the XML document. It defines rules to validate the value of elements and attributes as well as their formats. Once a schema is defined, a Schema Validator (For example: XmlValidatingReader class of .NET xml library, SQL Server 2005, etc.) can validate an XML document against the rules defined in the Schema.

Schema Languages

As the usage of XML increased, schema languages were also developed to support the validation requirements. DTD, XDR, SOX, Schematron, DSD, DCD, DDML, RELAX NG are a few among them. We will have a quick glance into DTD and XDR in this chapter. An introduction to the other Schema languages is beyond the scope of this book.

Document Type Definition (DTD)

Document Type Definitions (DTD) is one of the commonly used methods for describing XML documents. A DTD can be used to define the basic structure of the XML instance, data type of the attributes, default and fixed values, etc. DTDs are relatively simple and have a compact syntax. On the other side, they have their own syntax. DTD does not provide ample support for common requirements like namespaces, data types, etc.

The following is an approximate representation of the DTD which describes the sample XML we saw previously.

```
<!ELEMENT Employee (Name)>
<!ELEMENT Name (First, Middle, Last)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Middle (#PCDATA)>
<!ELEMENT Last (#PCDATA)>
```
1– Introduction to XML schema

An XML document may have a reference to an external DTD file or can have the DTD embedded as part of the XML file. The XML document given below has embedded DTD information.

```xml
<?xml version="1.0"?>
<!DOCTYPE Employee [  
<!ELEMENT Employee  (Name)>  
<!ELEMENT Name      (First, Middle, Last)>  
<!ELEMENT First     (#PCDATA)>  
<!ELEMENT Middle    (#PCDATA)>  
<!ELEMENT Last      (#PCDATA)>  
]

<Employee>
  <Name>
    <First>Jacob</First>
    <Middle>V</Middle>
    <Last>Sebastian</Last>
  </Name>
</Employee>
```

The example given below shows an XML document that refers to an external DTD file.

```xml
<?xml version="1.0"?>
<!DOCTYPE Employee SYSTEM "employee.dtd">
<Employee>
  <Name>
    <First>Jacob</First>
    <Middle>V</Middle>
    <Last>Sebastian</Last>
  </Name>
</Employee>
```

**XML-Data Reduced (XDR)**

XML-Data Reduced (XDR) was developed in 1998 with the joint effort of Microsoft and University of Edinburgh. The syntax of XDR is very close to that of XSD and is documented at:

http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm.

Microsoft implemented XDR in MSXML Parser. SQL Server 2000 supported creating XML using Annotated XDR Schemas. In SQLXML 4.0 Microsoft added support for XSD schemas and deprecated XDR schemas.

An approximate XDR representation of the sample schema (We have seen an XSD version as well as DTD version) is the following:

```xml
<Schema name="Employee" xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="First"/>
  <ElementType name="Middle"/>
</Schema>
```
XML Support in SQL Server 2000

SQL Server 2000 was released with a basic set of XML programming capabilities, which includes generating XML data using FOR XML and reading XML data with OPENXML.

**FOR XML**

FOR XML helps to generate XML output from the results of a TSQL query. When used with AUTO, RAW or EXPLICIT, FOR XML provides different levels of control over the structure of the XML result being generated.

**OPENXML**

OPENXML() function shreds an XML document and provides a rowset representation of the XML data.

**SQLXML**

SQLXML is an add-on which added additional XML capabilities to SQL Server 2000. Before you could access any of those features, SQLXML should be configured in IIS using the MMS snap-in which is installed as part of SQLXML setup.

With the assistance of SQLXML, SQL Server 2000 offered the following additional features:

**Querying Data over HTTP**

Once SQLXML is configured in IIS, you can send a TSQL statement over HTTP to the server and receive the results.
XML Views

An XML View provides an XML representation of the relational data of one or more tables. Using an XML View, you can run XPath queries on the relational data exposed by the XML View. XML views can be used with Updategrams to perform updates on the database.

Web Services

Another important feature exposed by SQLXML is the capability to expose SQL Server 2000 as a web service. This will enable you to send HTTP SOAP requests to the server to execute stored procedures, functions, etc.

XML Support in SQL Server 2005

In addition to many enhancements to the existing XML features, SQL Server 2005 introduced a new data type: XML. Let us briefly examine the XML capabilities of SQL Server 2005.

FOR XML – To generate XML Data

SQL Server 2000 supported three different modes with FOR XML, namely: RAW, AUTO and EXPLICIT. SQL Server 2005 added a new mode, PATH. The usage of PATH is relatively simple and it helps to achieve many of the complex XML formatting requirements which were possible only with complex usage of EXPLICIT earlier.

XML Data Type

SQL Server 2005 introduced a new data type: XML. An instance of the XML data type represents an XML document or fragment. XML data type can be used to define columns and can also be passed as parameters to functions and stored procedures. Functions can return XML values. You can declare XML variables in TSQL.

XQuery Support

The support for XML data type raised the requirement for querying the XML document stored in an XML column or variable. SQL Server 2005 supports XQuery (XML Query Language). XQuery is a W3C specification designed to provide a flexible and standardized way of querying XML data.
Support for XSD (XML Schema Definition)

SQL Server 2005 supports XSD (XML Schema Definition) to perform validations on the structure and value of XML documents. XML columns and variables can be bound to an XSD schema and the Schema Processing Engine will perform validations on the data, based on the schema definition. Please note that the support of XSD in SQL Server 2005 is still limited.

XML Support Enhancements in SQL Server 2008

SQL Server 2008 added several enhancements to the XML capabilities of the previous version of SQL server.

Schema Validation Enhancements

SQL Server 2008 added a number of enhancements to Schema Validation. Let us quickly examine them.

Lax Validation Support

To increase the flexibility of an XSD schema, wild card components are often used. This is usually done by using elements <xsd:any> or <xsd:anyAttribute>. Wild card components allow adding content that is not known at the time of schema design.

SQL Server 2005 always had options to either "skip" the validation of such elements or to perform a "strict" validation. When validation is "skipped" no validation is applied on such elements. When validation is set to "strict" the elements are always validated.

SQL Server 2008 supports "lax" validation, which validates only elements and attributes for which schema declarations are available. If the schema declaration is not available, the validation will be skipped for those elements and attributes."lax" validation is explained in Chapter 13

Full support for date, time and dateTime data types

XSD specification defines time-zone information as optional with date, time and dateTime data types. However, the XSD implementation of SQL Server 2005 required time zone information to be present with a date, time or dateTime value. However, it did not preserve the time zone information. The value is normalized into UTC date/time.
SQL Server 2008 removes this limitation. You can omit time zone information when storing date, time or dateTime data types. If you include time-zone information, the information is preserved. We will see these enhancements in Chapter 7.

**Improved support for union and list types**

SQL Server 2008 adds support for list types that contains union types. It allows union types that contain list types as well. We will examine this in Chapter 7.

**XQuery Enhancements**

SQL Server 2008 adds support for the "let" clause in the "query()" method of the XML data type. Refer to Books Online for a detailed explanation of the "let" clause.

**XML DML Enhancements**

The only significant DML change is the support for inserting an XML variable (or value of XML type) into another XML variable or XML column (using the XQuery "modify()" method with "insert" operation).

**TYPED and UNTYPED XML**

SQL Server 2005/2008 supports two flavors of XML known as TYPED and UNTYPED. Typed XML is associated with an XML Schema that defines the structure of the XML variable or column. Any text data can be stored to an UNTYPED XML column or variable as long as it is in XML format. But a TYPED XML column or variable must strictly follow the structure defined in the XML schema (XSD).

TYPED XML has many advantages over UNTYPED XML.

- SQL Server has prior knowledge about a TYPED XML column or variable because it is bound to a schema known to it. This knowledge will help the query optimizer generate better query plans.
- When a TYPED XML is used, SQL Server knows the data types of elements and attributes and can do better query processing.
- SQL Server can perform validations when value is inserted or updated. If the XML document or fragment does not pass all the validations defined in the XML Schema, SQL Server will raise an error and will not modify/insert the data.

By using an XSD schema, you can perform all sorts of validations that need to be done before accepting the XML data. If you work with XML data often you may be familiar with the following requirements, which will make your application less prone to error.
Validate the structure of the XML

Example:

<address> should occur after <name>. <phone> is optional but there should be one or more <item> elements.

Validate the data types

Example:

<zip> should be numeric, <age> should be numeric, <phone> is alpha numeric, <dateOfBirth> should be a valid date value, <maritalStatus> should be Boolean.

Perform restrictions on values

Example:

<hiredate> should not be earlier than 1900. <age> should be between 18 and 80. <itemnumber> should have 3 digits, followed by a "-" and then 4 alpha-numeric characters.

There are many more validations that we might need to do, depending upon the nature of our application and the type of data that we receive. Performing such validations without the help of a SCHEMA will be extremely difficult most of the time. Think of reading/parsing the XML document using your favorite XML library and validating each element and attribute. Though you could do this for some of the basic validations, most of the real life validations will be impractical to perform without a SCHEMA.

By using an XSD schema you can define all the validation rules using simple XML structure, and SQL Server 2005 will perform all the validations on your behalf.
How This Book Is Organized

Note: XSD support is added with SQL Server 2005 and that is the subject of this entire book. The XSD support added by SQL Server 2005 is further enhanced by SQL Server 2008. Whenever I say SQL Server, I would be referring to both SQL Server 2005 and 2008. I will use the version number to refer to a specific version of SQL Server.

Chapter 1: This chapter gives an introduction to XML schema, discusses a few different schema languages, and then explains the XML support extended by SQL Server 2000, 2005 and 2008.

Chapter 2: In this chapter we will write our first XSD schema. We will then have a look at XML namespaces and become familiar with SQL Server XML Schema collections. We will see how to validate XML instances against schema collections and will have a quick overview of the SSMS XSD editor.

Chapter 3: This chapter presents a fictitious company, North Pole Corporation, which needs some web services developed for their order processing application. This chapter identifies the structure of the XML data to be exchanged and determines the validations to be performed using XSD.

Chapter 4: This chapter explains the basic building blocks of XSD. We will see Element Declarations, Attribute Declarations, Simple Types, Complex Types, Attribute Groups, Element Groups, Order Indicators, Occurrence Indicators, annotations, etc. This chapter also explains how to associate data types with element declarations and perform additional validations on the value of elements and attributes.

Chapter 5: This chapter explains element declarations in detail. We will examine global and local element declarations and discuss each attribute that an element declaration can take. We will examine name, id, type, default, fixed, nillable, substitutionGroup, abstract, block, final, minOccurs, maxOccurs, ref and form attributes.

Chapter 6: This chapter explains attribute declarations in detail. We will have a quick glance into the behavior of elements and attributes and will discuss global and local attribute declarations. We will then look at each parameter of an attribute declaration and will discuss attribute groups and their usages.

Chapter 7: This chapter starts with a basic discussion on the importance of data types. We will discuss the characteristics of XSD data types and XSD Primitive Data Types in much detail.

Chapter 8: This chapter focuses on Simple Types. It explains the difference between simple types and complex types and discusses local and global simple type declarations. We will examine simple type derivation as well as the enhancements added to list and union types in SQL Server 2008. We will also examine how to restrict derivation of simple types.
Chapter 9: We will discuss XSD Derived Data Types in this chapter. This chapter starts with a discussion on XSD Primitive Data types and Derived data types and then discusses the facets of the primitive data types. Finally, it explains each of the Derived Data Types in detail.

Chapter 10: We will examine Complex Types in this chapter. We will examine local and global complex types and discuss the different content models of complex types. We will also examine order indicators, occurrence indicators and element groups.

Chapter 11: This chapter focuses on derivation of complex types. It discusses the derivation of each content model by restriction as well as by extension. It also explains how to control complex type derivation.

Chapter 12: This chapter explains the Regular Expression language supported in XSD. We will discuss regular expressions, patterns, meta characters, case sensitivity, shorthand character classes, negative expressions and character class subtraction.

Chapter 13: This chapter discusses some advanced XSD concepts. We will discuss element wildcards, attribute wildcards, and extension of wildcard elements and attributes. We will see different ways a schema processor validates wildcard declarations. We will then see the different attributes of a schema declaration.

Chapter 14: This chapter looks closer into XML Schema Collections. We will discuss how to create schema collections having more than one schema definition. We will then discuss how to alter schema collections, how to retrieve the definition of a schema collection from SQL Server and will examine the various system catalog views related to schema collections.