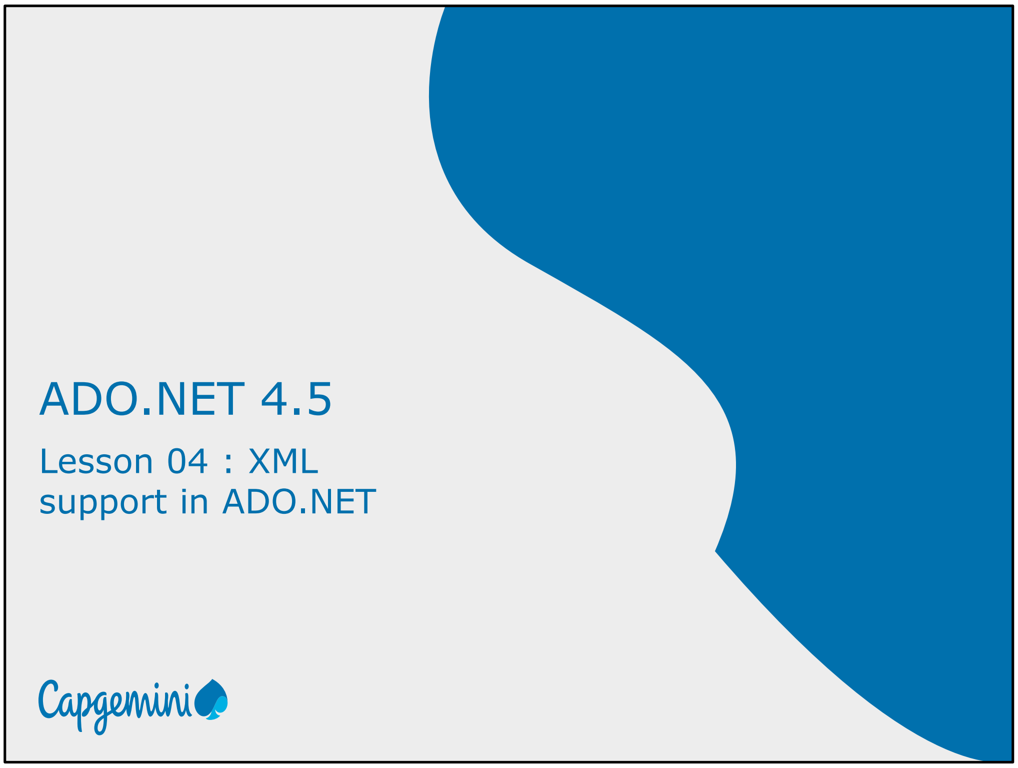
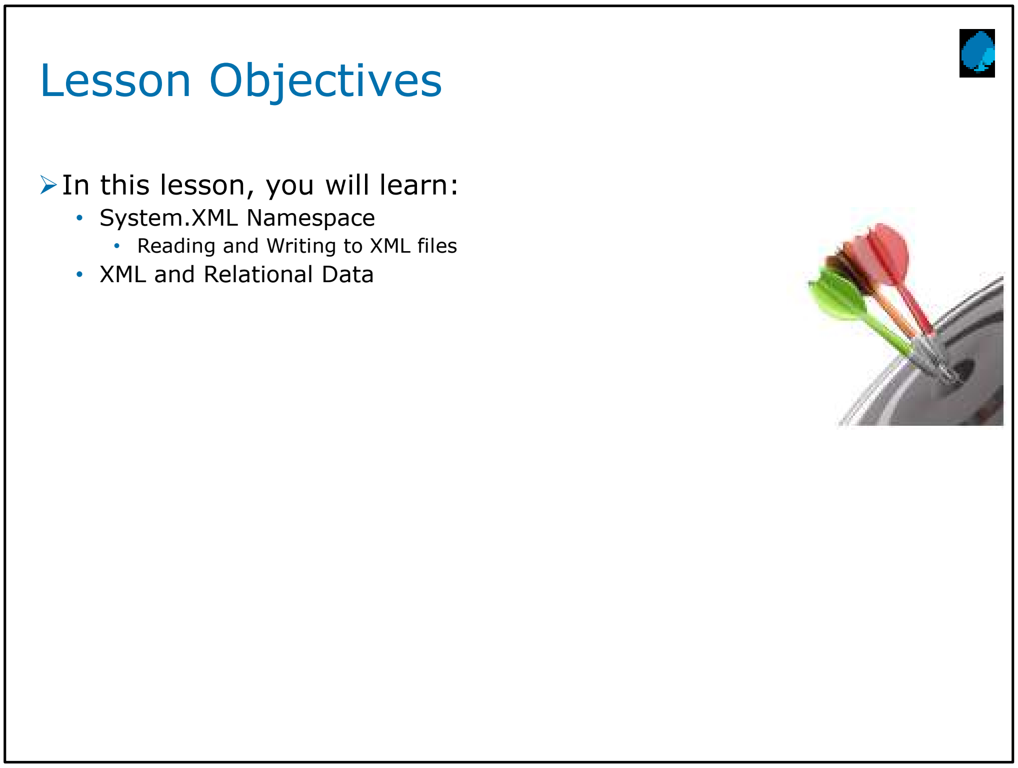


ADO.NET 4.5

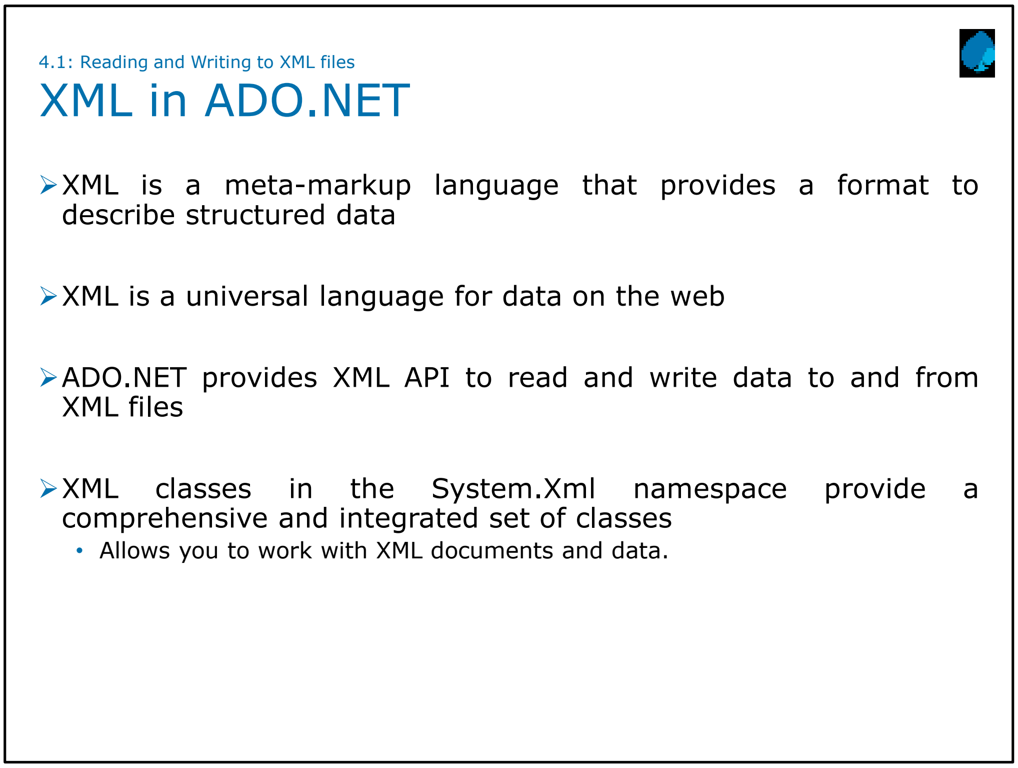
Lesson 04 : XML support in ADO.NET

Capgemini



Lesson Objectives

- In this lesson, you will learn:
  - System.XML Namespace
    - Reading and Writing to XML files
  - XML and Relational Data



4.1: Reading and Writing to XML files

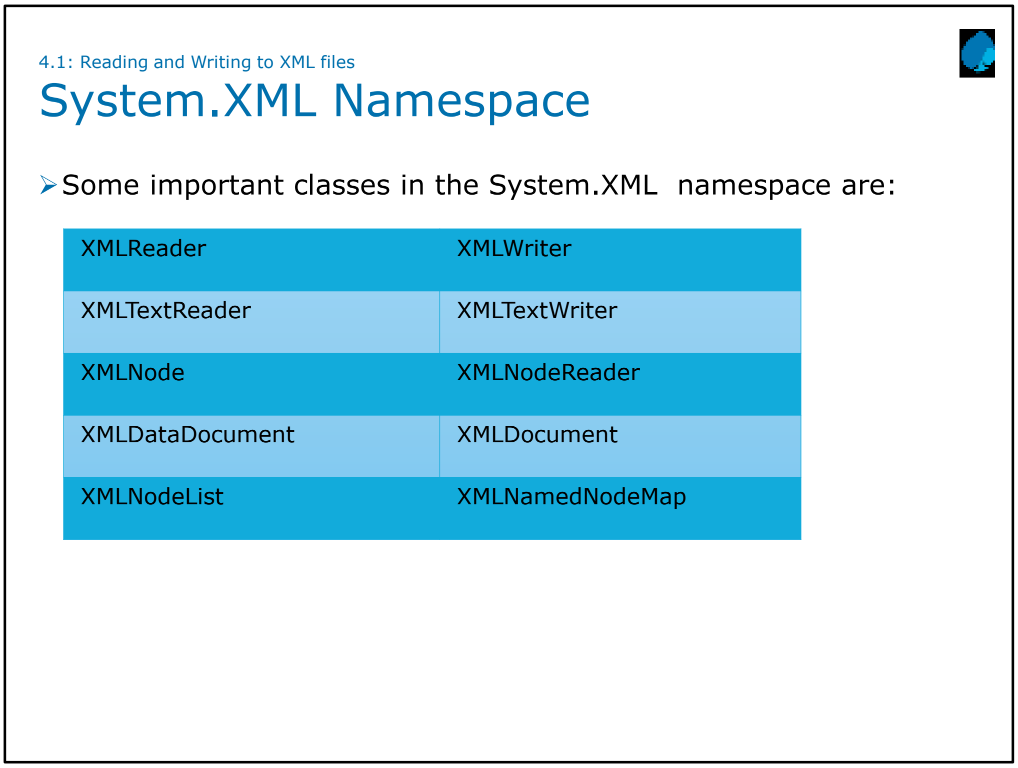
XML in ADO.NET

- XML is a meta-markup language that provides a format to describe structured data
- XML is a universal language for data on the web
- ADO.NET provides XML API to read and write data to and from XML files
- XML classes in the System.Xml namespace provide a comprehensive and integrated set of classes
  - Allows you to work with XML documents and data.

Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data. XML enables a new generation of Web-based data viewing and manipulation applications. XML is the universal language for data on the Web. XML gives developers the power to deliver structured data from a wide variety of applications to the desktop for local computation and presentation. It is used extensively in applications that are written by using general-purpose languages like C# or VB.NET.
It is mostly used for the following:

to exchange data between applications,
to store configuration information,
to persist temporary data,
as a base for generating web pages or reports, and
for many other things.

The XML API provided through System.Xml namespace provide a comprehensive and integrated set of classes, allowing you to work with XML documents and data. This namespace has a comprehensive set of XML classes for parsing, validation, and manipulation of XML data using readers, writers, and World Wide Web Consortium (W3C) DOM-compliant components. It also covers XML Path Language (XPath) queries and Extensible Stylesheet Language Transformations (XSLT). You should note that the XML namespace allows you to get similar results in a number of different ways .



4.1: Reading and Writing to XML files

System.XML Namespace

- Some important classes in the System.XML namespace are:

XMLReader	XMLWriter
XMLTextReader	XMLTextWriter
XMLNode	XMLNodeReader
XMLDataDocument	XMLDocument
XMLNodeList	XMLNamedNodeMap

Support for processing XML is provided by the classes in the System.Xml namespace in .NET. Some of the important classes are:

XMLReader - An abstract reader class that provides fast, non-cached XML data. XmlReader is forward-only.

XMLWriter - Represents an abstract writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

XMLTextReader – This class extends XmlReader and conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations. XmlTextReader provides the following functionality:

    Enforces the rules of well-formed XML.
    XmlTextReader does not provide data validation.
    Checks that DocumentType nodes are well-formed. XmlTextReader checks the DTD for well-formedness, but does not validate using the DTD.
    Does not expand default attributes.

XMLTextWriter - This class extends the XMLWriter. Represents a writer that provides a fast, non-cached, forward-only way of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations. XmlTextWriter does not check for the following:

- **Invalid characters** in attribute and element names.
- **Unicode characters** that do not fit the specified encoding. If the Unicode characters do not fit the specified encoding, the XmlTextWriter does not escape the Unicode characters into character entities.
- **Duplicate attributes** Characters in the DOCTYPE public identifier or system identifier.

- **XMLNode** – An abstract class that represents a single node in an XML document. XmlNode is the base class in the .NET implementation of the DOM.

- **XMLNodeReader** – This class represents a reader that provides fast, non-cached forward only access to XML data in an XmlNode and extends the XMLReader**.** The XmlNodeReader has the ability to read an XML DOM subtree. This class does not support DTD or schema validation.

- **XMLDocument** – Represents an XML document and it extends XMLNode. It provides a tree representation in memory of an XML document, enabling navigation and editing.

- **XMLDataDocument** – This class extends the XMLDocument. Allows structured data to be stored, retrieved, and manipulated through a relational DataSet. Allows the mixing of XML and relational data in the same view.

- **XMLNodeList** – To represents an ordered collection of nodes you can use XMLNodeList. The XmlNodeList collection reflects changes to the children of the node object that it was created from immediately in the nodes returned by the XmlNodeList properties and methods. XmlNodeList supports iteration and indexed access.

  - **XmlNodeList** is returned by the following properties and methods.
  - **XmlNode.ChildNodes** Returns a XmlNodeList containing all the children of the node.
  - **XmlNode.SelectNodes** XmlNodeList is returned containing a collection of nodes matching the XPath query.
  - **GetElementsByTagName** Returns XmlNodeList containing a list of all descendant elements that match the specified name. This method is available in both the XmlDocument and XmlElement classes.

- **XMLNamedNodeMap** – This class represents a collection of nodes that can be accessed by name or index. XmlNamedNodeMap is returned by the following three properties.
  - **XmlElement.Attributes** Returns XmlAttributeCollection, a class which inherits from XmlNamedNodeMap.
  - **XmlDocumentType.Entities** Returns an XmlNamedNodeMap containing XmlEntity objects. The XmlNamedNodeMap is read-only.
  - **XmlDocumentType.Notations** Returns an XmlNamedNodeMap containing XmlNotation objects. The XmlNamedNodeMap is read-only.

4.1: Reading and Writing to XML files

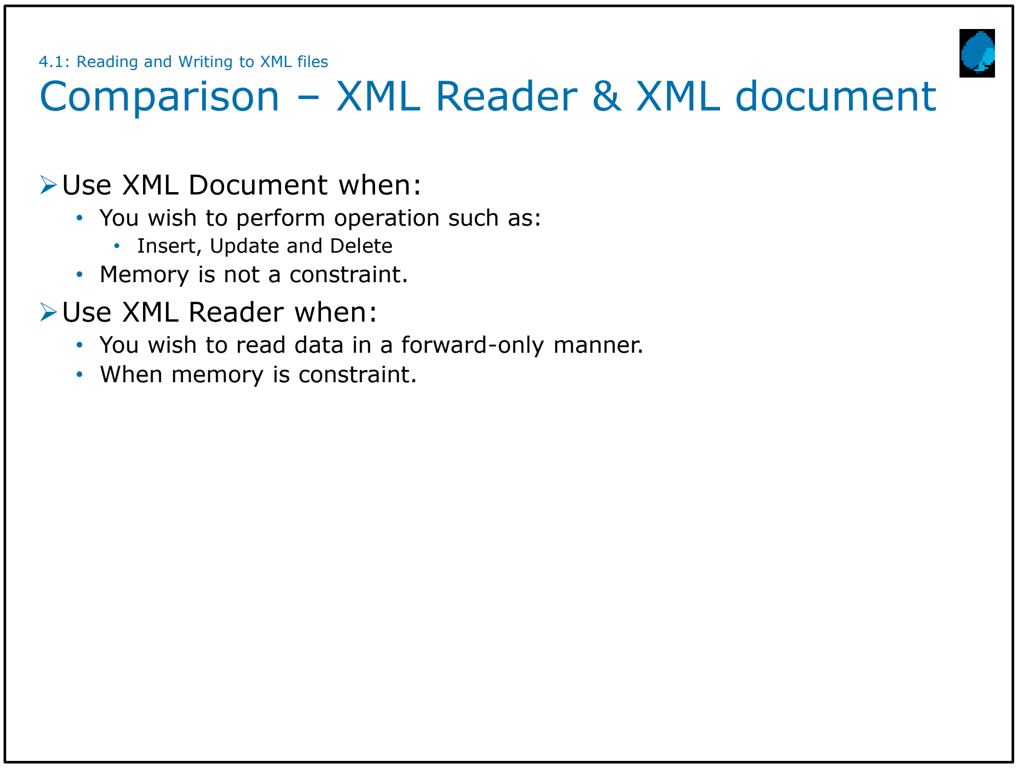
Demo

- Using the XML Text Reader

4.1: Reading and Writing to XML files

Demo

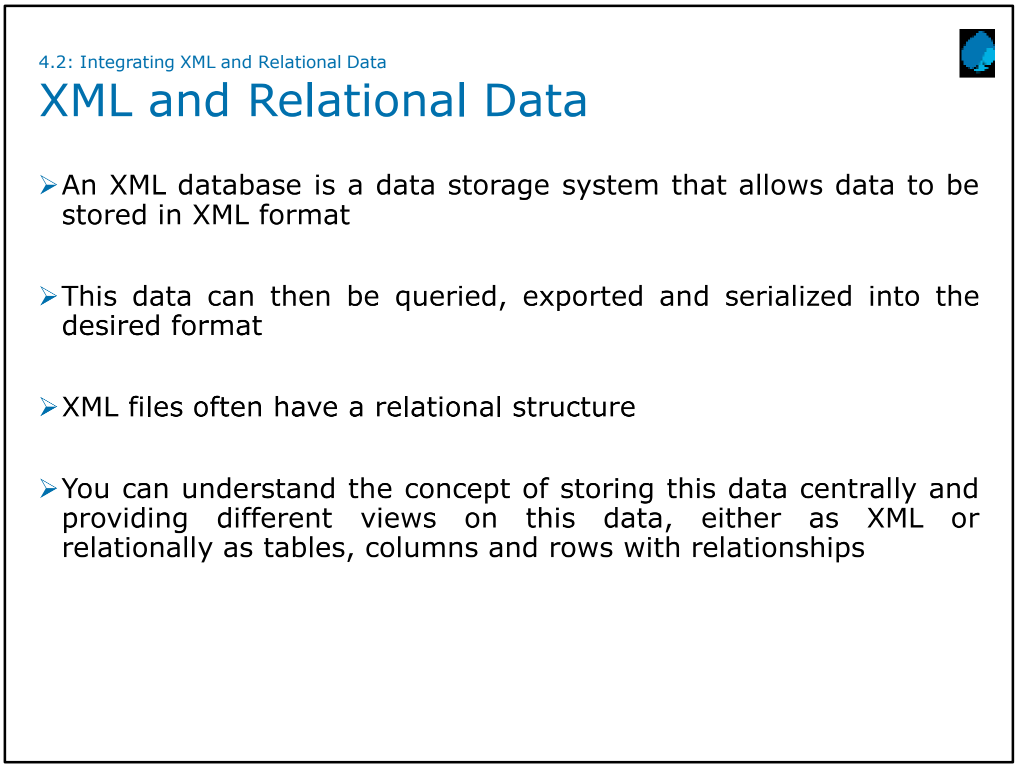
- Using the XML Document



4.1: Reading and Writing to XML files

Comparison – XML Reader & XML document

- Use XML Document when:
  - You wish to perform operation such as:
    - Insert, Update and Delete
  - Memory is not a constraint.
- Use XML Reader when:
  - You wish to read data in a forward-only manner.
  - When memory is constraint.



4.2: Integrating XML and Relational Data

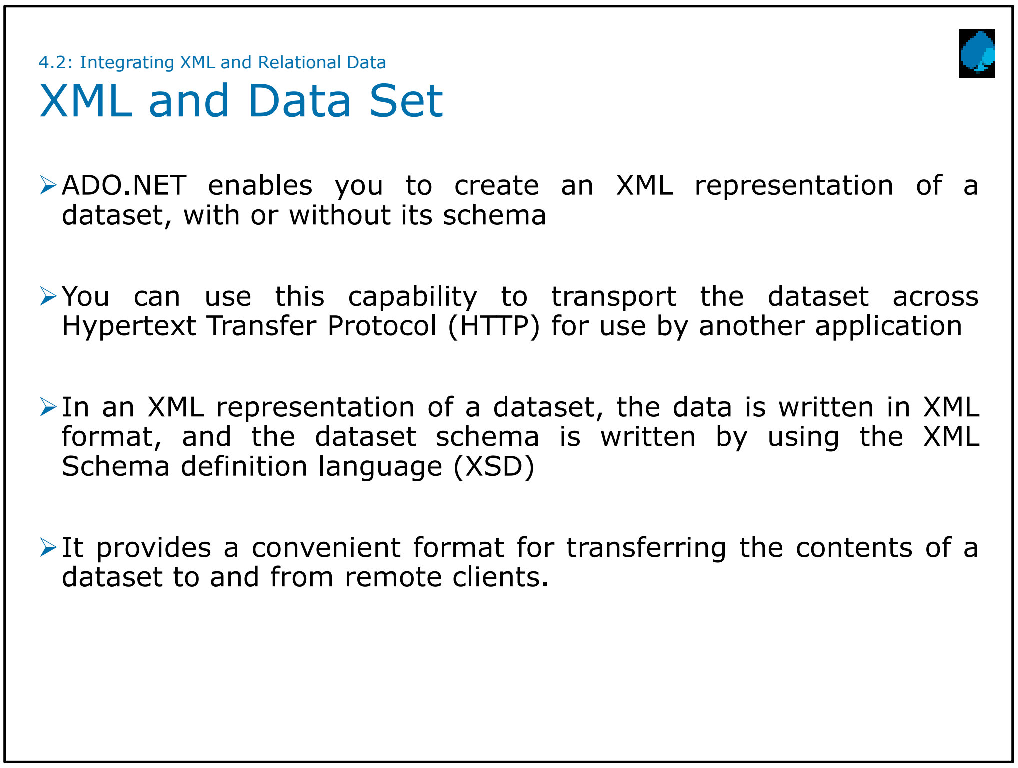
XML and Relational Data

- An XML database is a data storage system that allows data to be stored in XML format
- This data can then be queried, exported and serialized into the desired format
- XML files often have a relational structure
- You can understand the concept of storing this data centrally and providing different views on this data, either as XML or relationally as tables, columns and rows with relationships

XML often has a relational structure (the fictional books in a bookstore for example), as well as being structured. This set of topics cover the concept of storing this data centrally and providing different views on this data, either as XML or relationally as tables, columns and rows with relationships. This disconnected store of data which, for example, could represent a business object in the middle tier that enforces business rules, could provide its data as XML to a browser via Extensible Stylesheet Language Transformations (XSLT), across the Internet to another Web site, or to a local application via the relational tables. However, the data gets amended, you can update that data to a database on a transaction basis.

A DataSet represents an in-memory cache of data as a collection of tables and relationships between those tables. It is, in effect, a locally cached database. This provides a disconnected cache of data, like a message, that enables dealing with chunks of data. The DataSet has no knowledge of where the data came from. It may have come from a file, a database connection, or from a stream. A DataSet provides a relational view onto this stored data.

The XmlDataDocument provides XML APIs for accessing this in-memory cache of data, as well as supporting reading and writing XML. The XmlDataDocument is a DataSet-aware object. Creation of an XmlDataDocument implicitly creates a DataSet (accessed as a property) that provides a relational view onto the data XML data. This symbiotic relationship between these two objects provides a powerful technique for accessing data either relationally or as XML, irrespective of the mechanism by which the data was sourced.

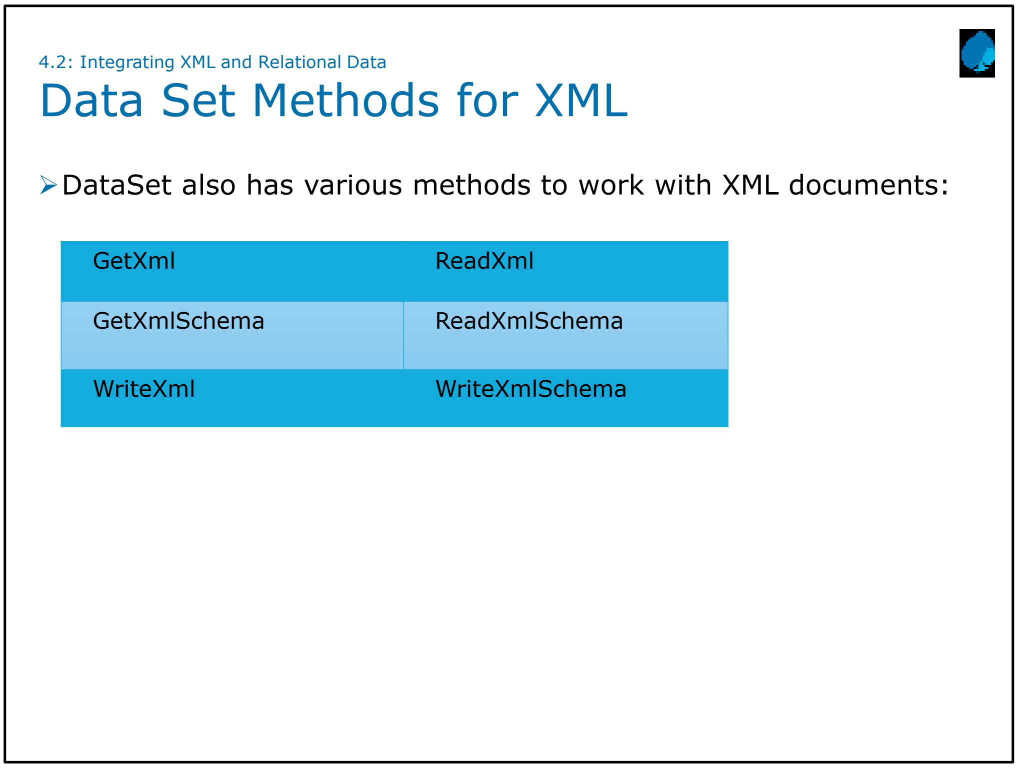


4.2: Integrating XML and Relational Data

XML and Data Set

- ADO.NET enables you to create an XML representation of a dataset, with or without its schema
- You can use this capability to transport the dataset across Hypertext Transfer Protocol (HTTP) for use by another application
- In an XML representation of a dataset, the data is written in XML format, and the dataset schema is written by using the XML Schema definition language (XSD)
- It provides a convenient format for transferring the contents of a dataset to and from remote clients.

With ADO.NET you can fill a DataSet from an XML stream or document. XmlDataDocument is a DataSet-aware object. Relationship between these two objects provides a powerful technique to access data either relationally or as XML, irrespective of the mechanism by which the data is sourced.
You can use the XML stream or document to supply to the DataSet either data, schema information, or both. The information supplied from the XML stream or document can be combined with existing data or schema information already present in the DataSet.
ADO.NET also allows you to create an XML representation of a DataSet, with or without its schema, in order to transport the DataSet across HTTP for use by another application or XML-enabled platform. In an XML representation of a DataSet, the data is written in XML and the schema, if it is included inline in the representation, is written using the XML Schema definition language (XSD). XML and XML Schema provide a convenient format for transferring the contents of a DataSet to and from remote clients.



4.2: Integrating XML and Relational Data

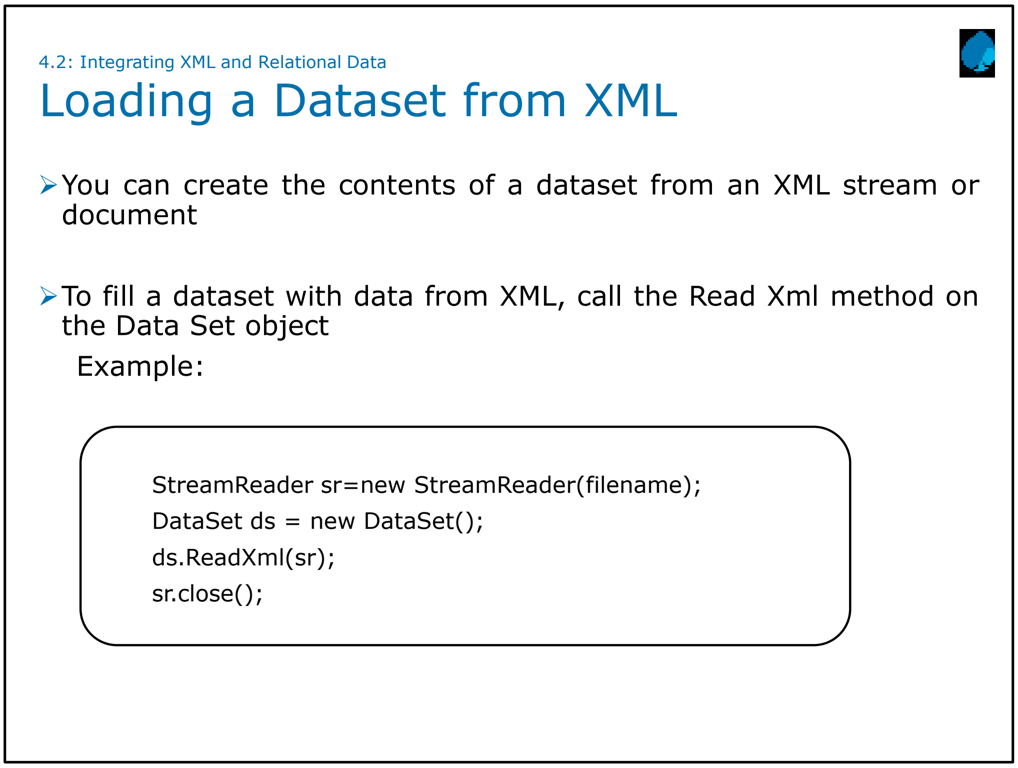
Data Set Methods for XML

- DataSet also has various methods to work with XML documents:

GetXml	ReadXml
GetXmlSchema	ReadXmlSchema
WriteXml	WriteXmlSchema

The DataSet is silently rebuilt as a binary and promptly usable object. The same serialization facilities are available to applications through a bunch of methods, a pair of which clearly stands out. The methods to work with XML Document offered by Dataset are:

GetXml: Returns a string value of the XML representation of the data stored in the DataSet.
GetXmlSchema: Returns the XML Schema for the XML representation of the data stored in the DataSet.
ReadXml: Populates a DataSet object with the specified XML data read from a stream or a file.
ReadXmlSchema: Loads the specified XML schema information into the current DataSet object.
WriteXml: Writes the XML data, and optionally the schema, that represents the DataSet. Can write to a stream or a file.
WriteXmlSchema: Writes the string being the XML schema information for the DataSet .Can write to a stream or a file.



4.2: Integrating XML and Relational Data

Loading a Dataset from XML

- You can create the contents of a dataset from an XML stream or document
- To fill a dataset with data from XML, call the Read Xml method on the Data Set object

  Example:

```
StreamReader sr=new StreamReader(filename);
DataSet ds = new DataSet();
ds.ReadXml(sr);
sr.close();
```

The contents of an ADO.NET DataSet can be created from an XML stream or document. With the .NET Framework you have great flexibility over what information is loaded from XML, and how the schema or relational structure of the DataSet is created.

To fill a DataSet with data from XML, use the ReadXml method of the DataSet object. The ReadXml method will read from a file, a stream, or an XmlReader, and takes as arguments the source of the XML plus an optional XmlReadMode argument. The ReadXml method reads the contents of the XML stream or document and loads the DataSet with data. The method creates the relational schema for the DataSet depending on the read mode specified, and whether or not a schema already exists in the DataSet. The following code snippet illustrates the typical code you would use to load a DataSet from XML:

```
StreamReader sr=new StreamReader(filename);
DataSet ds = new DataSet();
ds.ReadXml(sr);
sr.close();
```

When loading the contents of XML sources into a DataSet, ReadXml does not merge rows whose primary key information match. To merge an existing DataSet with one loaded from XML, you first have to create a new DataSet, and then merge the two using the Merge method. During the merging, the rows that get overwritten are those with matching primary keys. An alternate way to merge existing DataSet objects with contents read from XML is through the DiffGram format.

Let us take a look at the various read modes the ReadXml supports:

- **Auto**: This is the default. It examines the XML and chooses the most suitable option in the following order:
    - If the XML is a DiffGram, DiffGram is used.
    - If the DataSet contains a schema or the XML contains an inline schema, ReadSchema is used.
    - If the DataSet does not contain a schema and the XML does not contain an inline schema, InferSchema is used.

- **ReadSchema**: Reads any inline schema and loads the data and schema. If the DataSet already contains a schema, new tables are added from the inline schema to the existing schema in the DataSet. If the DataSet does not contain a schema, and there is no inline schema and no data is read.
- **IgnoreSchema**: Ignores any inline schema and loads the data into the existing DataSet schema. Any data that does not match the existing schema is discarded. If no schema exists in the DataSet, no data is loaded.
- **InferSchema**: Ignores any inline schema and infers the schema per the structure of the XML data, then loads the data.
- **DiffGram**: Reads a DiffGram and adds the data to the current schema. DiffGram merges new rows with existing rows where the unique identifier values match.
- **Fragment**: It continues reading multiple XML fragments until the end of the stream is reached. Fragments that match the DataSet schema are appended to the appropriate tables. The fragments that do not match the DataSet schema are discarded.

4.2: Integrating XML and Relational Data

Writing Dataset to XML Document

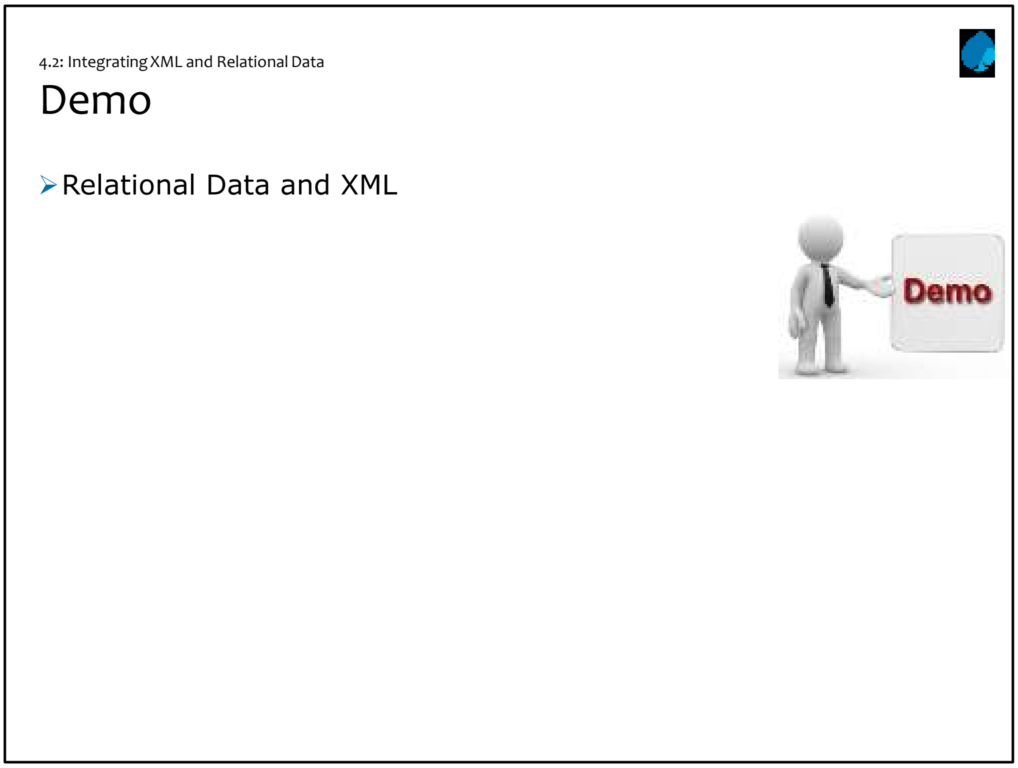
- You can write an XML representation of a dataset, with or without its schema, by calling the Write Xml method on the dataset object.
- When a dataset is written as XML data, the rows in the dataset are written in their current versions.

Example:

```
SqlDataAdapter daCustomer = new SqlDataAdapter("Select *
from Customers", con);
DataSet ds = new DataSet();
da.Fill(ds, "Customers");
ds.WriteXml("C:\\CustomerXML.xml");
```

In ADO.NET you can write an XML representation of a DataSet, with or without its schema. If schema information is included inline with the XML, it is written using the XML Schema definition language (XSD). The schema contains the table definitions of the DataSet as well as the relation and constraint definitions.
When a DataSet is written as XML data, the rows in the DataSet are written in their current versions. However, the DataSet can also be written as a DiffGram so that both the current and the original values of the rows is included.
The XML representation of the DataSet can be written to a file, a stream, an XmlWriter, or a string. These choices provide great flexibility for how you transport the XML representation of the DataSet. To obtain the XML representation of the DataSet as a string, use the GetXml method.
To write a DataSet to a file, stream, or XmlWriter, use the WriteXml method. The first parameter you pass to WriteXml is the destination of the XML output. For example, pass a string containing a file name, a System.IO.TextWriter object, and so on. You can pass an optional second parameter of an XmlWriteMode to specify how the XML output is to be written.

Write modes are as follows:
IgnoreSchema: Writes the current contents of the DataSet as XML data, without an XML Schema.
WriteSchema: Writes current contents of the DataSet as XML data with the relational structure, as inline XML Schema.
DiffGram: Writes the entire DataSet as a DiffGram, including original and current values.


4.2: Integrating XML and Relational Data

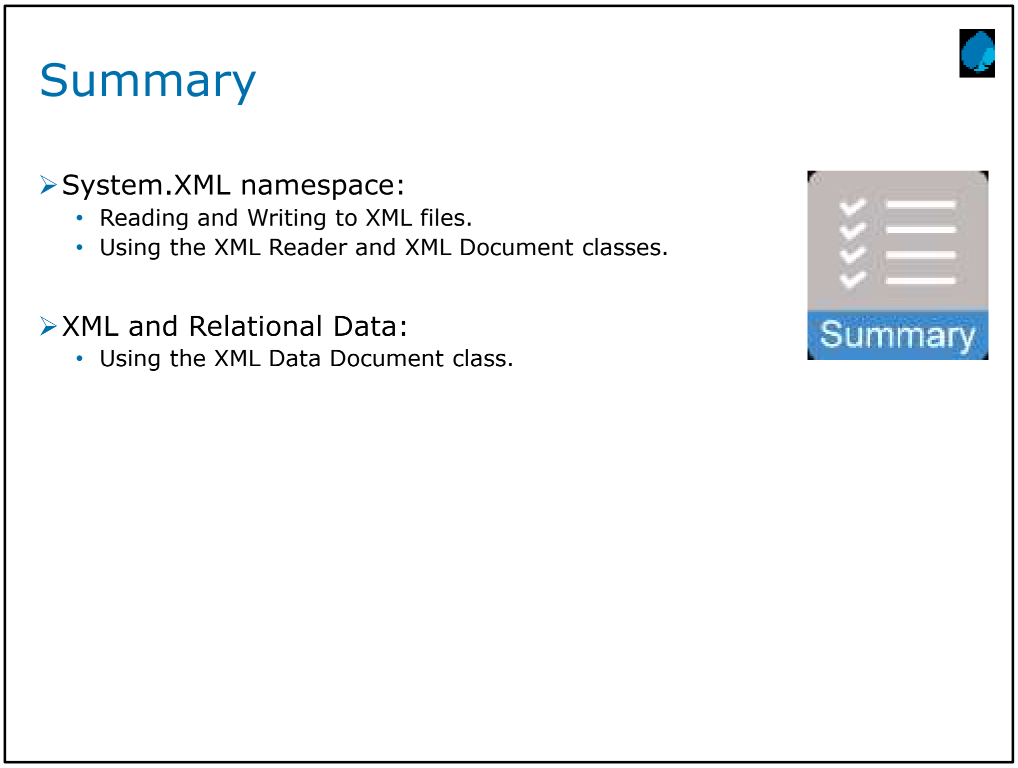
Demo

- Relational Data and XML

When writing an XML representation of a DataSet that contains DataRelation objects, you will most likely want the resulting XML to have the child rows of each relation nested within their related parent elements. To accomplish this, set the Nested property of the DataRelation to true when you add the DataRelation to the DataSet.

Following code snippet shows how to use WriteXml:

```
System.IO.StreamWriter xmlSW = new
System.IO.StreamWriter("Customers.xml");
custDS.WriteXml(xmlSW, XmlWriteMode.WriteSchema);
xmlSW.Close();
```

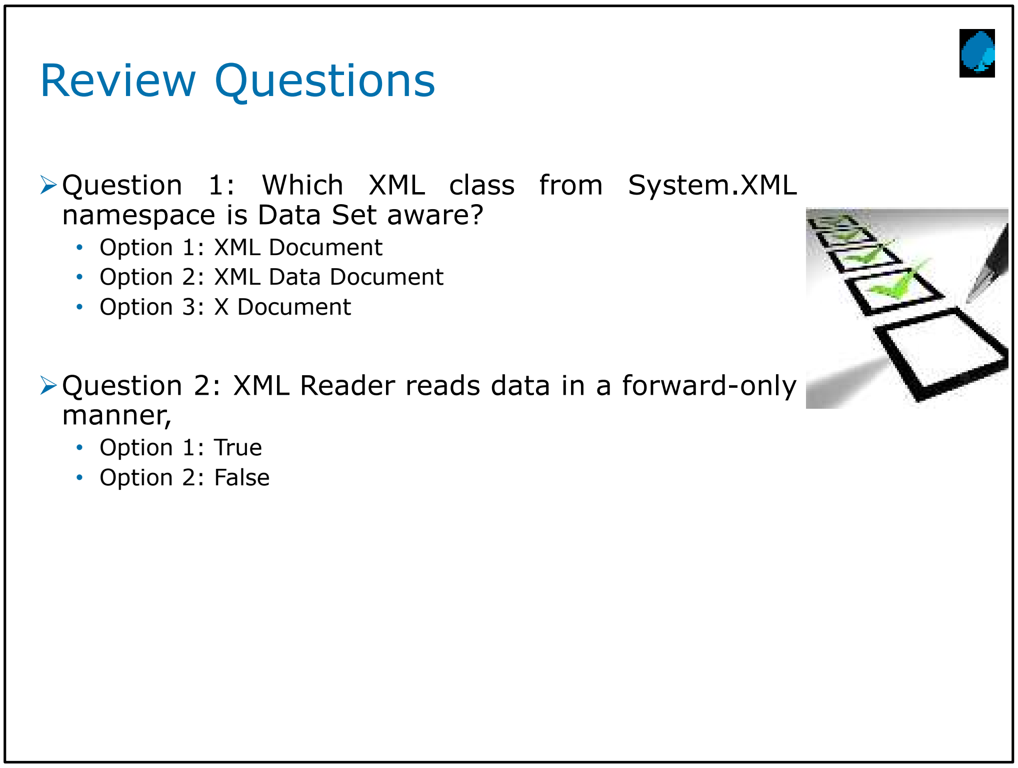


Summary

- System.XML namespace:
  - Reading and Writing to XML files.
  - Using the XML Reader and XML Document classes.
- XML and Relational Data:
  - Using the XML Data Document class.

Summary

Add the notes here.



Review Questions

- Question 1: Which XML class from System.XML namespace is Data Set aware?
  - Option 1: XML Document
  - Option 2: XML Data Document
  - Option 3: X Document
- Question 2: XML Reader reads data in a forward-only manner,
  - Option 1: True
  - Option 2: False

Add the notes here.

Review Questions

- Question 3: The _________ class allows you to directly go to a particular element without worrying about the root element.

Add the notes here.