CHƯƠNG 2
VẤN ĐỀ I/O TRONG .NET

NỘI DUNG

- Giới thiệu
- Streams
  - Streams cho tập tin
  - Encoding data
  - Stream cho dữ liệu nhị phân và text
  - Serialization
  - Deserialization
  - Xuất một cơ sở dữ liệu dùng stream

- I/O là vấn đề rất quan trọng đối với truyền thông trên mạng
- Khảo sát các hoạt động I/O bên dưới
- Khảo sát vấn đề stream để phục vụ cho việc chuyển đổi các đối tượng phức tạp sang stream

STREAMS

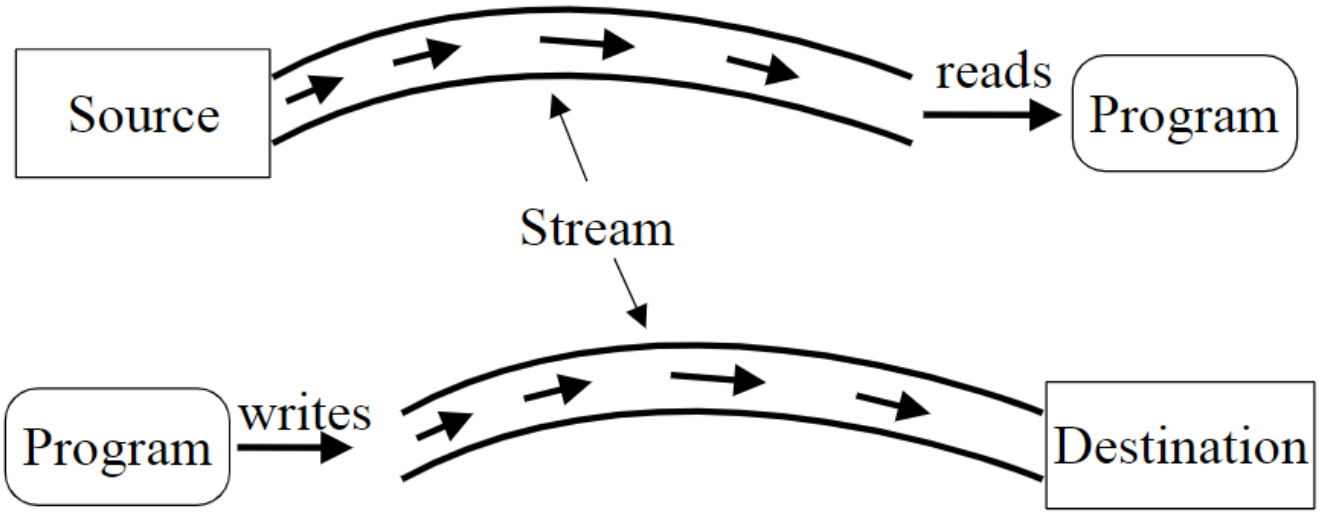
- Kiến trúc dựa trên stream đã được phát triển trong .NET
- Các thiết bị I/O bao gồm từ máy in, đĩa cứng cho đến card mạng
- Không phải các thiết bị đều có chức năng giống nhau → stream cũng không hỗ trợ các phương thức giống nhau
- canRead(), canSeek(), canWrite() chỉ khả năng stream ứng với thiết bị cụ thể

STREAMS

- Dữ liệu được truyền theo hai hướng

- Đọc dữ liệu: đọc dữ liệu từ bên ngoài vào chương trình.

- Ghi dữ liệu: đưa dữ liệu từ chương trình ra bên ngoài.



STREAMS

- Hai stream quan trọng: **networkStream** và **fileStream**
- Hai cách dùng stream: **đồng bộ** và **bất đồng bộ**
- Khi dùng đồng bộ: luồng (thread) tương ứng sẽ tạm ngưng đến khi tác vụ hoàn thành hoặc lỗi
- Khi dùng bất đồng bộ: luồng (thread) tương ứng sẽ ngay tức thì quay về phương thức gọi nó và bất cứ lúc nào tác vụ hoàn thành sẽ có dấu hiệu chỉ thị, hoặc lỗi xảy ra

STREAMS

- Kiểu chương trình "treo" để chờ tác vụ hoàn thành không "thân thiện" cho lắm, do đó phương thức gọi đồng bộ phải dùng một luồng riêng
- Bằng cách dùng các luồng và phương thức gọi bất đồng bộ làm cho có cảm giác máy tính có thể làm được nhiều việc cùng lúc. Thực tế, hầu hết máy tính chỉ có 1 CPU, nên điều trên đạt được là do chuyển giữa các tác vụ trong khoảng một vài milliseconds

STREAMS CHO CÁC FILE

- Khởi tạo một ứng dụng .NET mới, thêm vào:
  - Một form
  - Một File Open Dialog control với tên openFileDialog
  - Một textbox với tên tbResults, lập thuộc tính multiline=true.
  - Hai buttons với tên btnReadAsync và btnReadSync

STREAMS CHO CÁC FILE

- Sử dụng namespace: using System.IO;
- Khai báo:

```
FileStream fs;
byte[] fileContents;
AsyncCallback callback;
delegate void InfoMessageDel(String info);
```

Khai báo thêm phương thức InfoMessageDel để tránh vấn đề tranh chấp bởi các thread tham chiếu đến một đối tượng trong lập trình mạng

STREAMS CHO CÁC FILE

- Thêm code xử lý biến cố Click của đối tượng btnReadAsync:

```
private void btnReadAsync_Click(object sender,
EventArgs e)
{
  openFileDialog.ShowDialog();
  callback = new AsyncCallback(fs_StateChanged);
  fs = new FileStream(openFileDialog.FileName,
FileMode.Open, FileAccess.Read, FileShare.Read,
4096, true);
  fileContents = new Byte[fs.Length];
  fs.BeginRead(fileContents, 0, (int)fs.Length,
callback,  null);
}
```

- Chú ý: Đọc 4096 byte/lần là cách hiệu quả nhất

STREAMS CHO CÁC FILE

- Nội dung hàm fs_StateChanged:

```
private void fs_StateChanged(IAsyncResult asyncResult)
{
    if (asyncResult.IsCompleted)
    {
        string s =
Encoding.UTF8.GetString(fileContents);
        InfoMessage(s);
        fs.Close();
    }
}
```

STREAMS CHO CÁC FILE

- Thêm code xử lý biến cố Click của đối tượng btnReadSync:

```
private void btnReadSync_Click(object sender,
EventArgs e)
{
    openFileDialog.ShowDialog();
    Thread thdSyncRead = new Thread(new
ThreadStart(syncRead));
    thdSyncRead.Start();
}
```

STREAMS CHO CÁC FILE

```
public void syncRead()
{
    FileStream fs;
    try
    {
        fs = new FileStream(openFileDialog.FileName,
FileMode.OpenOrCreate);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
```

STREAMS CHO CÁC FILE

```
fs.Seek(0, SeekOrigin.Begin);
    byte[] fileContents = new
byte[fs.Length];
    fs.Read(fileContents, 0, (int)fs.Length);
    string s =
Encoding.UTF8.GetString(fileContents);
    InfoMessage(s);
    fs.Close();
}
```

STREAMS CHO CÁC FILE

```
public void InfoMessage(String info)
{
    if (tbResults.InvokeRequired)
    {
        InfoMessageDel method = new
InfoMessageDel(InfoMessage);
        tbResults.Invoke(method, new object[] {
info });
        return;
    }
    tbResults.Text = info;
}
```

FILESTREAM

Phương thức hoặc thuộc tính	Mục đích
Constructor	Khởi tạo một thực thể mới của FileStream
Length	Độ dài của file, giá trị kiểu long
Position	Lấy ra hoặc thiết lập vị trí của con trỏ file, giá trị kiểu long
BeginRead()	Bắt đầu đọc bất đồng bộ
BeginWrite()	Bắt đầu ghi bất đồng bộ
Write()	Ghi một khối byte vào stream dùng dữ liệu trong bộ đệm
Read()	Đọc một khối byte từ stream và ghi vào trong bộ đệm
Lock()	Ngăn cản việc các tiến trình khác truy xuất vào tất cả hoặc một phần của file

ENCODING DATA

- Trong ví dụ trước ta đã dùng Encoding.UTF8.GetString() để chuyển đổi một mảng byte thành string.
- Các dạng hợp lệ là Unicode (Encoding.Unicode), ASCII, UTF7
- UTF8 dùng 1 byte cho một ký tự, Unicode dùng 2 byte cho mỗi ký tự

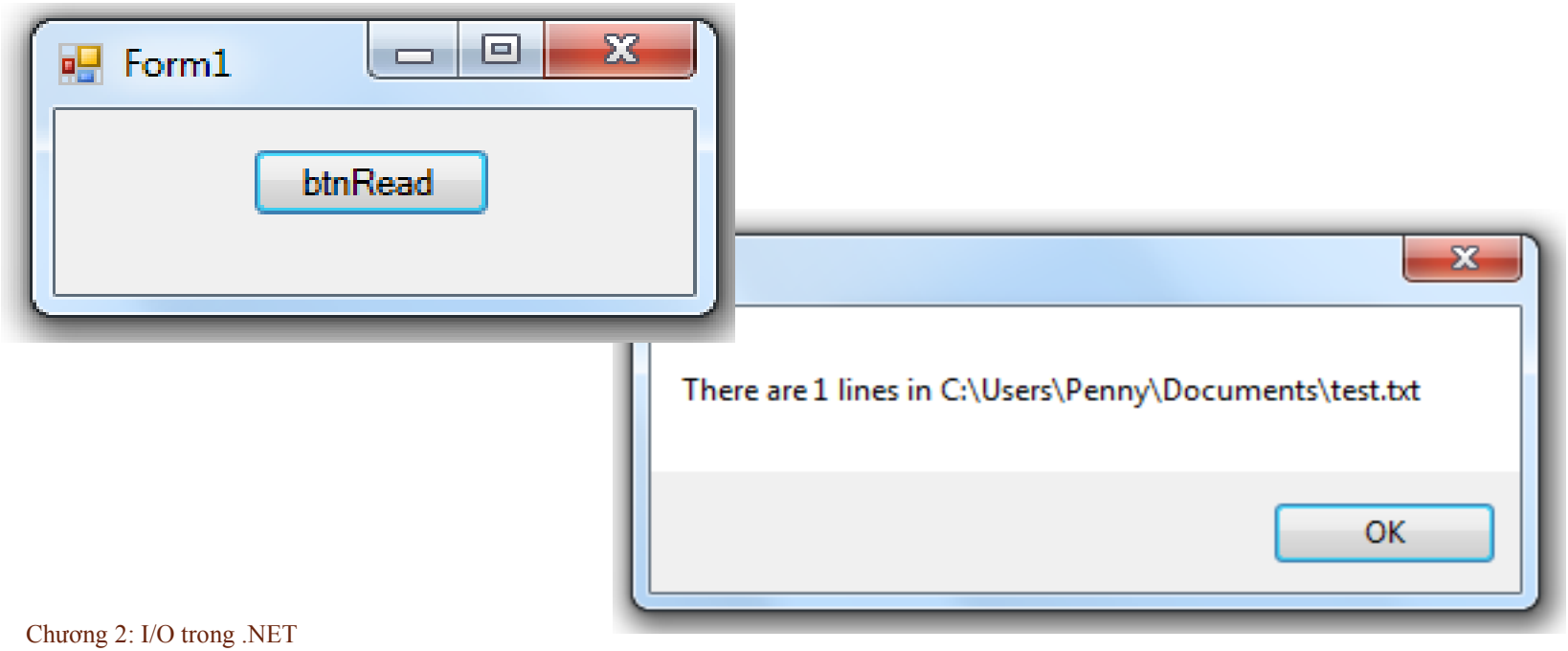
BINARY VÀ TEXT STREAMS

- Plain text là dạng thức phổ biến dùng trong các stream để con người dễ đọc và soạn thảo. Tương lai sẽ thay bằng XML.
- Đặc tính chung của plain text là mỗi đơn vị thông tin được kết thúc với mã phím enter (tổ hợp hai mã UTF8 là 10 và 13 trong C# hay vbCrLf trong VB.NET)

STREAMREADER

- **Chương trình đếm số dòng trong file**

Khi nhấn vào button btnRead sẽ đếm và thông báo số dòng có trong một tập tin bất kỳ.



STREAMREADER

```
private void btnRead_Click(object sender, System.EventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.ShowDialog();
    FileStream fs = new FileStream(ofd.FileName, FileMode.OpenOrCreate);
    StreamReader sr = new StreamReader(fs);
    int lineCount = 0;
    while (sr.ReadLine() != null)
    {
        lineCount++;
    }
    fs.Close();
    MessageBox.Show("There are " + lineCount + " lines in " + ofd.FileName);
}
```

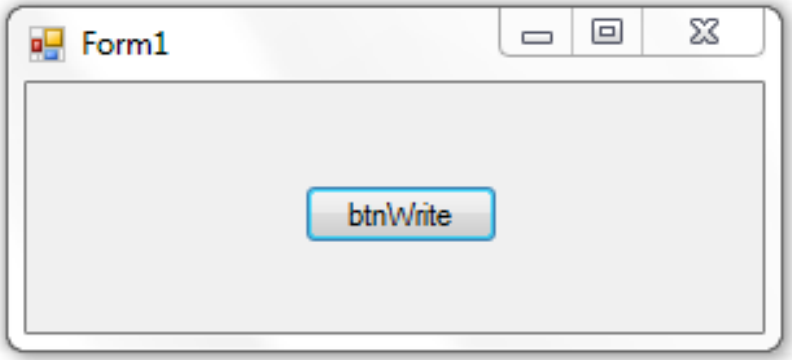
STREAMREADER

Phương thức hoặc Thuộc tính	Mục đích
Constructor	Khởi tạo một thực thể mới của StreamReader
Peek	Trả về ký tự kế tiếp, hoặc giá trị -1 nếu đến cuối stream
Read	Đọc ký tự kế tiếp hoặc một tập các ký tự từ input stream
ReadBlock	Đọc các ký tự từ stream hiện hành và ghi dữ liệu vào bộ đệm, bắt đầu tại vị trí chỉ định
ReadLine	Đọc một dòng ký tự từ stream hiện hành và trả về dưới dạng string
ReadToEnd	Đọc từ vị trí hiện hành đến cuối stream.

BINARYWRITER

- **Chương trình ghi file nhị phân**

Chương trình ghi thành file nhị phân với nội dung



BINARYWRITER

```
private void btnWrite_Click(object sender, EventArgs e)
{
      SaveFileDialog sfd = new SaveFileDialog();
      sfd.ShowDialog();
      FileStream fs = new FileStream(sfd.FileName,  FileMode.CreateNew);
      BinaryWriter bw = new BinaryWriter(fs);
      int[] myArray = new int[1000];
      for (int i = 0; i < 1000; i++)
      {
          myArray[i] = i;
          bw.Write(myArray[i]);
      }
      bw.Close();
 }
```

BINARYWRITER

Phương thức hoặc thuộc tính	Mục đích
Constructor	Khởi tạo một thực thể mới của BinaryWriter
Close	Đóng BinaryWriter hiện hành và stream liên quan
Seek	Định vị trí con trỏ trên stream hiện hành
Write	Ghi giá trị vào stream hiện hành
Write7BitEncodedInt	Ghi giá trị số nguyên 32 bit (dạng nén) vào stream hiện hành

SERIALIZATION

- Serialization là tiến trình mà một đối tượng .NET dùng để chuyển đổi vào trong một stream, do vậy dễ dàng truyền trên mạng cũng như ghi vào đĩa
- Tiến trình ngược lại được gọi là deserialization

SERIALIZATION

- Ví dụ điển hình là hệ thống đặt hàng, vốn có yêu cầu độ an toàn rất cao, vì vậy mỗi lỗi xảy ra phải được theo vết chặt chẽ
- Để đặt đơn hàng vào stream (trên đĩa hoặc trên mạng) ta có thể ghi mỗi giá trị dưới dạng text, dùng ký tự phân cách,… để xuất và tái tạo các đối tượng. Tuy nhiên cách dễ dàng nhất là dùng Serialization

SERIALIZATION

```
public enum purchaseOrderStates
{
   ISSUED,   DELIVERED,   INVOICED,   PAID
}
[Serializable()]
public class company
{
   public string name;
   public string address;
   public string phone;
}
[Serializable()]
```

SERIALIZATION

```
public class lineItem
{
   public string description;
   public int quantity;
   public double cost;
}
[Serializable()]
public class purchaseOrder
{
   private purchaseOrderStates _purchaseOrderStatus;
```

SERIALIZATION

```
private DateTime _issuanceDate;
private DateTime _deliveryDate;
private DateTime _invoiceDate;
private DateTime _paymentDate;
public company buyer;
public company vendor;
public string reference;
public lineItem[] items;
```

SERIALIZATION

```
public purchaseOrder()
{
   _purchaseOrderStatus = purchaseOrderStates.ISSUED;
   _issuanceDate = DateTime.Now;
}
public void recordDelivery()
{
   if (_purchaseOrderStatus == purchaseOrderStates.ISSUED)
   {
      _purchaseOrderStatus = purchaseOrderStates.DELIVERED;
      _deliveryDate = DateTime.Now;
   }
}
```

SERIALIZATION DÙNG SOAPFORMATTER

```
company Vendor = new company();
company Buyer = new company();
lineItem Goods = new lineItem();
purchaseOrder po = new purchaseOrder();
Vendor.name = "Acme Inc.";
Buyer.name = "Wiley E. Coyote";
Goods.description = "anti-RoadRunner cannon";
Goods.quantity = 1;
Goods.cost = 599.99;
```

SERIALIZATION DÙNG SOAPFORMATTER

```
po.items = new lineItem[1];
po.items[0] = Goods;
po.buyer = Buyer;
po.vendor = Vendor;
SoapFormatter sf = new SoapFormatter();
FileStream fs = File.Create("..\\po.xml");
sf.Serialize(fs, po);
fs.Close();
```

DESERIALIZATION DÙNG SOAPFORMATTER

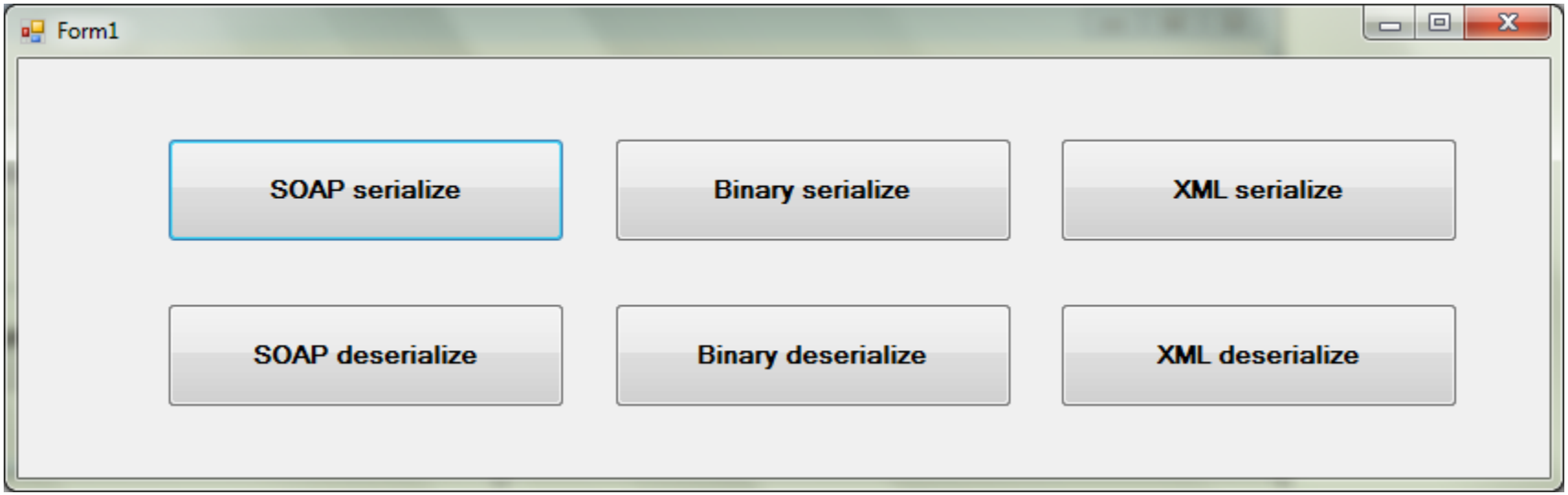
```
SoapFormatter sf = new SoapFormatter();
FileStream fs = File.OpenRead("..\\po.xml");
purchaseOrder po = (purchaseOrder)sf.Deserialize(fs);
fs.Close();
MessageBox.Show("Customer is " + po.buyer.name +
   "\nVendor is " + po.vendor.name + ", phone is " +
po.vendor.phone +
            "\nItem is " + po.items[0].description + " has
quantity " +
            po.items[0].quantity.ToString() + ", has cost " +
po.items[0].cost.ToString());
```

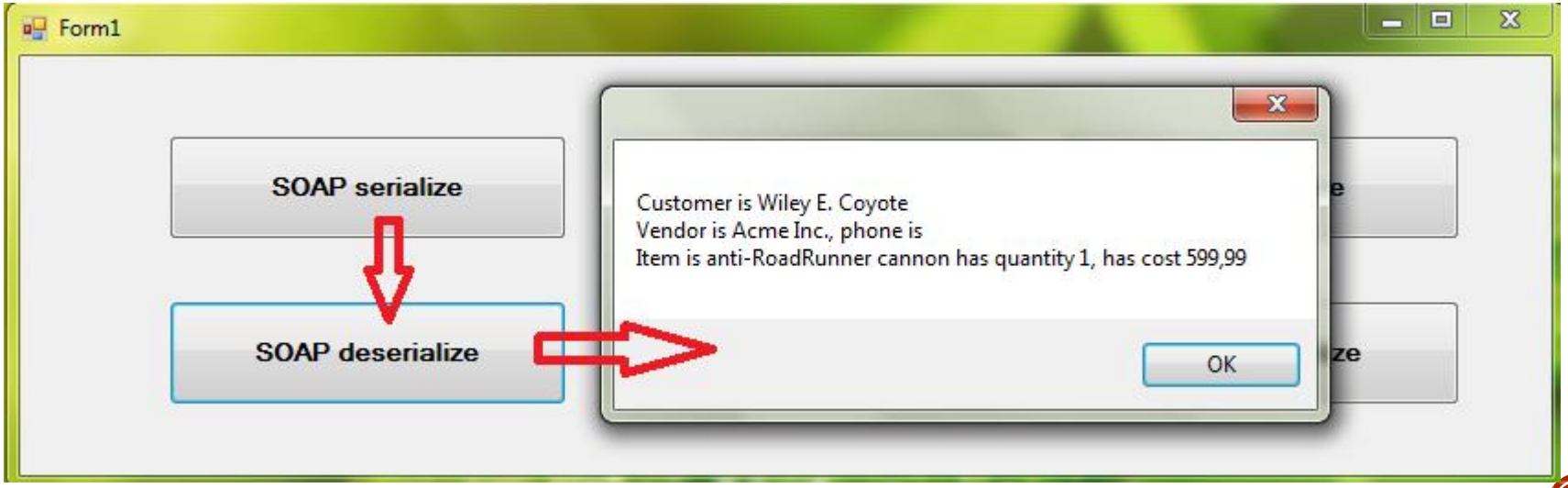
SOAPFORMATTER

Simple Object Access Protocol (SOAP)

Phương thức hoặc thuộc tính	Mục đích
Constructor	Khởi tạo một thực thể mới của SoapFormatter
Deserialize	Deserialize một stream thành một đối tượng, đồ thị
Serialize	Serialize một đối tượng hoặc một đồ thị liên kết với các đối tượng
AssemblyFormat	Lấy ra hoặc thiết lập định dạng, trong đó các assembly names được serialize
TypeFormat	Lấy ra hoặc thiết lập định dạng, trong đó các type descriptions được cấu trúc sẵn trong stream được serialize
TopObject	Lấy ra hoặc thiết lập ISoapMessage trong đó đối tượng nằm trên SOAP được deserialize

KẾT QUẢ MINH HỌA





SERIALIZATION DÙNG BINARYFORMATTER

- Định dạng của SOAP tương đối ấn tượng, tuy nhiên không gọn nhẹ nên khá tiêu tốn băng thông đường truyền
- Trong trường hợp ấy, phương pháp khả thi hơn là BinaryFormatter

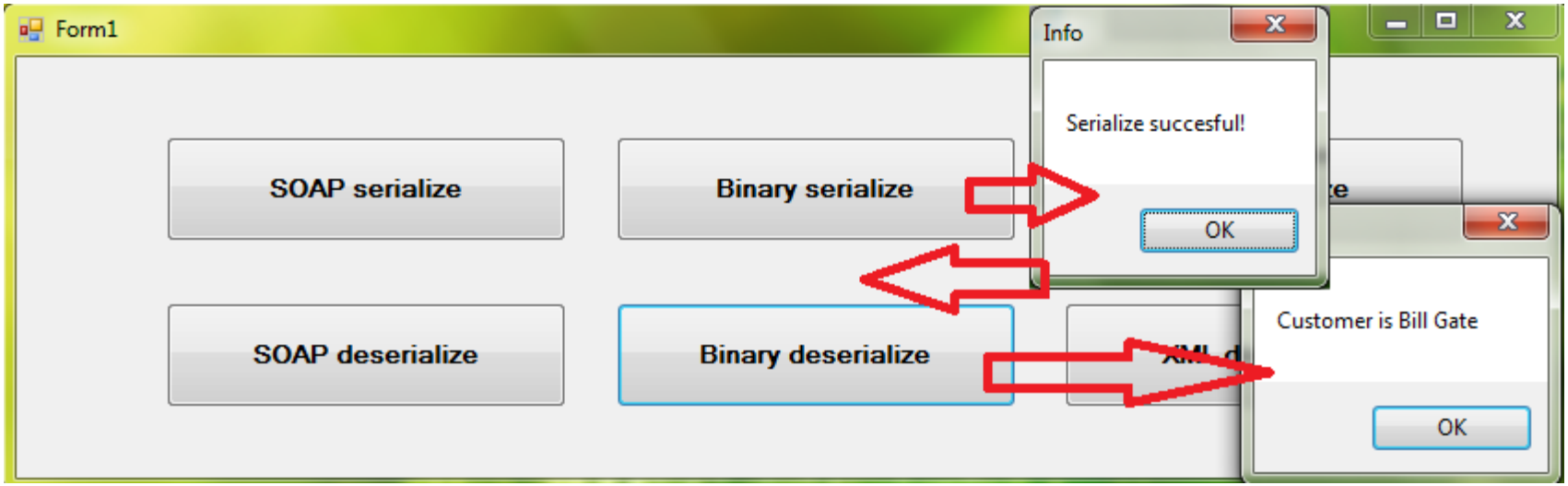
SERIALIZATION DÙNG BINARYFORMATTER

```
company Vendor = new company();
company Buyer = new company();
lineItem Goods = new lineItem();
//tương tự ví dụ SoapFormatter
 BinaryFormatter bf = new BinaryFormatter();
FileStream fs = File.Create("..\\po_bin.txt");
bf.Serialize(fs, po);
fs.Close();
MessageBox.Show("Serialize succesful!", "Info");
```

DESERIALIZATION DÙNG BINARYFORMATTER

```
BinaryFormatter bf = new BinaryFormatter();
FileStream fs = File.OpenRead("..\\po_bin.txt");
purchaseOrder po = (purchaseOrder)bf.Deserialize(fs);
fs.Close();
MessageBox.Show("Customer is " + po.buyer.name);
```

KẾT QUẢ MINH HỌA



SHALLOW SERIALIZATION

- Bất kỳ khi nào một đối tượng được serialized không có các thành viên private và protected thì được gọi là Shallow serialization
- Tuy nhiên phương pháp này đôi khi gây ra sự sao chép sai các đối tượng, không thể giải quyết việc tham chiếu vòng tròn giữa các đối tượng

SHALLOW SERIALIZATION

- Thuận lợi của nó là sử dụng XML schema definition (XSD) để định nghĩa các kiểu
- Các chuẩn XSD bảo đảm thể hiện chính xác trên mọi nền tảng
- SOAP formatter chỉ dùng trên hệ thống dạng CLR và không được chuẩn hóa trên các nền tảng không phải là .NET

SERIALIZATION DÙNG XMLSERIALIZER

```
company Vendor = new company();
company Buyer = new company();
lineItem Goods = new lineItem();
//tương tự ví dụ SoapFormatter
XmlSerializer xs = new XmlSerializer(po.GetType());
FileStream fs = File.Create("..\\po1.xml");
xs.Serialize(fs, po);
fs.Close();
```

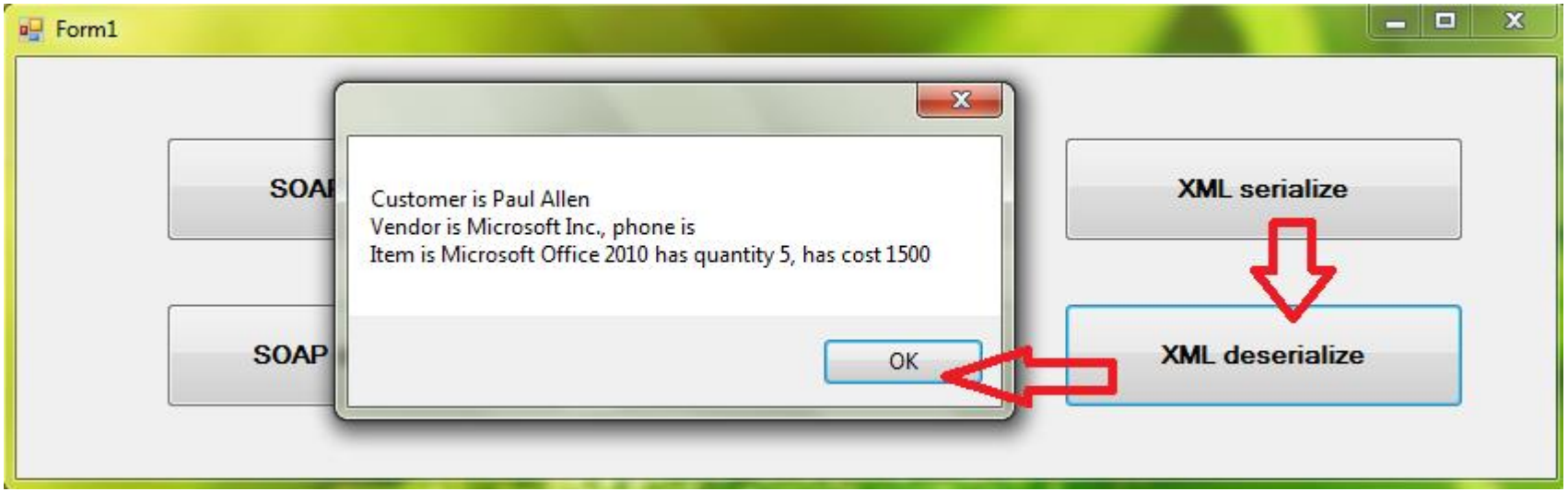
DESERIALIZATION DÙNG XMLSERIALIZER

```
purchaseOrder po = new purchaseOrder();
XmlSerializer xs = new XmlSerializer(po.GetType());
FileStream fs = File.OpenRead("..\\po1.xml");
po = (purchaseOrder)xs.Deserialize(fs);
fs.Close();
MessageBox.Show("Customer is " + po.buyer.name +
    "\nVendor is " + po.vendor.name + ", phone is " +
po.vendor.phone +   "\nItem is " + po.items[0].description +
" has quantity " +
    po.items[0].quantity.ToString() + ", has cost " +
po.items[0].cost.ToString());
```

XMLSERIALIZER

Phương thức hoặc thuộc tính	Mục đích
Constructor	Khởi tạo một thực thể mới của XmlSerializer
Deserialize	Deserialize một tài liệu XML
Serialize	Serialize một đối tượng thành tài liệu XML
FromTypes	Trả về một mảng các đối tượng XmlSerializer được tạo từ một mảng các kiểu
CanDeserialize	Lấy ra giá trị cho biết XmlSerializer có thể deserialize một tài liệu XML nào đó được hay không.

KẾT QUẢ MINH HỌA



GHI MỘT DATABASE VÀO STREAM

- Hầu hết các ứng dụng thương mại dùng cơ sở dữ liệu để lưu trữ dữ liệu của họ
- Để truyền dữ liệu trên mạng, chúng phải được ghi vào stream. Cách dễ dàng nhất là serialize dataset đó
- Chú ý: SQL Server và Oracle cung cấp cơ chế truy xuất trực tiếp vào cơ sở dữ liệu của nó và phải dùng trước để serialize

TỔNG QUAN LẬP TRÌNH CƠ SỞ DỮ LIỆU

- Có 2 chủ đề tập trung trong phần này:
  - Chuỗi kết nối: chỉ vị trí và kiểu của dữ liệu
  - Các phát biểu SQL: mô tả hoạt động đối với dữ liệu
- Cần phải dùng namespace System.Data.OleDb cho cơ sở dữ liệu Access và System.Data.SqlClient cho cơ sở dữ liệu SQL Server

CHUỖI KẾT NỐI

Kiểu Database	Chuỗi kết nối
Microsoft Access	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=<đường dẫn>\<.mdb file>
SQL Server	Data Source=<tên máy chủ>/<địa chỉ IP>; Initial Catalog=<tên database >; User ID=<user>; Password=<password>; Persist Security Info=True/False; Integrated security=True/False; *Trong đó: Integrated security=True thì sử dụng Windows Authentication* *Persist security info: Thiết lập mặc định cho persist security info là false. Thiết lập sang giá trị true sẽ cho phép các dữ liệu nhạy cảm bao gồm UserID và password có thể đọc được khi kết nối mở*

KẾT NỐI CƠ SỞ DỮ LIỆU: VÍ DỤ 1

```
private static string strCon;
public static SqlConnection cn;
strCon = "Data Source=" + txtTenSerVer.Text + ";Initial
Catalog=qlhokhau;Persist Security Info=True;User
ID=sa;Password=123456";
try {
    cn = new SqlConnection(strCon);
    cn.Open();
}
```

KẾT NỐI CƠ SỞ DỮ LIỆU: VÍ DỤ 1

```
catch (Exception e)
{
    MessageBox.Show("Chi tiết kỹ thuật: " + e.ToString(),
"Thông báo lỗi kết nối dữ liệu!!!", MessageBoxButtons.OK,
MessageBoxIcon.Error);
return;
};
```

KẾT NỐI CƠ SỞ DỮ LIỆU: VÍ DỤ 2

```
private static string strCon;

public static OleDbConnection cn;

string myConnection =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +

dataPath + ";Persist Security Info=True;Jet OLEDB:Database
Password=123456";

try {

   cn = new OleDbConnection(strCon);

   cn.Open();

}
```

KẾT NỐI CƠ SỞ DỮ LIỆU: VÍ DỤ 2

```
catch (Exception e)
{
    MessageBox.Show("Chi tiết kỹ thuật: " + e.ToString(),
"Thông báo lỗi kết nối dữ liệu!!!", MessageBoxButtons.OK,
MessageBoxIcon.Error);
return;
};
```

BỐN THAO TÁC CHÍNH

- Đọc dữ liệu ra: lệnh Select
- Thêm các dòng dữ liệu mới vào: lệnh Insert
- Xóa bỏ các dòng dữ liệu: lệnh Delete
- Cập nhật thông tin cho các dòng đã có: lệnh Update

VÍ DỤ CÁC THAO TÁC

- Lệnh Select tổng quát có dạng:
  Select * from table where column=‘điều kiện lọc’
- Lệnh Update tổng quát có dạng:
  Update table set column=’dữ liệu mới’ where column=‘điều kiện lọc’
- Lệnh Delete tổng quát có dạng:
  Delete from table where column=‘điều kiện lọc’
- Lệnh Insert tổng quát có dạng:
  Insert into table (column) values (’dữ liệu mới’)

VÍ DỤ CÁC THAO TÁC

- Để thực thi các thao tác Update, Delete, Insert ta phải gọi phương thức ExecuteNonQuery():

```
public void nonQuery(string szSQL, string szDSN)
{
  OleDbConnection DSN = new
OleDbConnection(szDSN);
  DSN.Open();
  OleDbCommand SQL = new OleDbCommand(SQL,
DSN);
   SQL.ExecuteNonQuery();
   DSN.Close();
}
```

VÍ DỤ CÁC THAO TÁC

- Thao tác Select (thực hiện trước khi serialize):

```
public DataSet Query(string szSQL, string szDSN)
{
  DataSet ds = new DataSet();
  OleDbConnection DSN = new OleDbConnection(szDSN);
  DSN.Open();
  OleDbCommand SQL = new OleDbCommand(szSQL,DSN);
  OleDbDataAdapter Adapter = new OleDbDataAdapter(SQL);
  Adapter.Fill(ds,"sql");
  DSN.Close();
  return(ds);
}
```

DATASET SERIALIZATION

- Giả sử cơ sở dữ liệu là Access và dùng cơ chế XML serialization, ta phải khai báo các namespace sau:

```
using System.Data.OleDb
using System.IO
using System.Xml.Serialization
```

- Thực hiện serialize cơ sở dữ liệu được tiến hành như sau:

DATASET SERIALIZATION

```
string szDSN = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\\purchaseOrder.mdb";

OleDbConnection DSN =  new OleDbConnection(szDSN);

XmlSerializer xs = new XmlSerializer(typeof(DataSet));

DataSet ds = new DataSet();

DSN.Open();
```

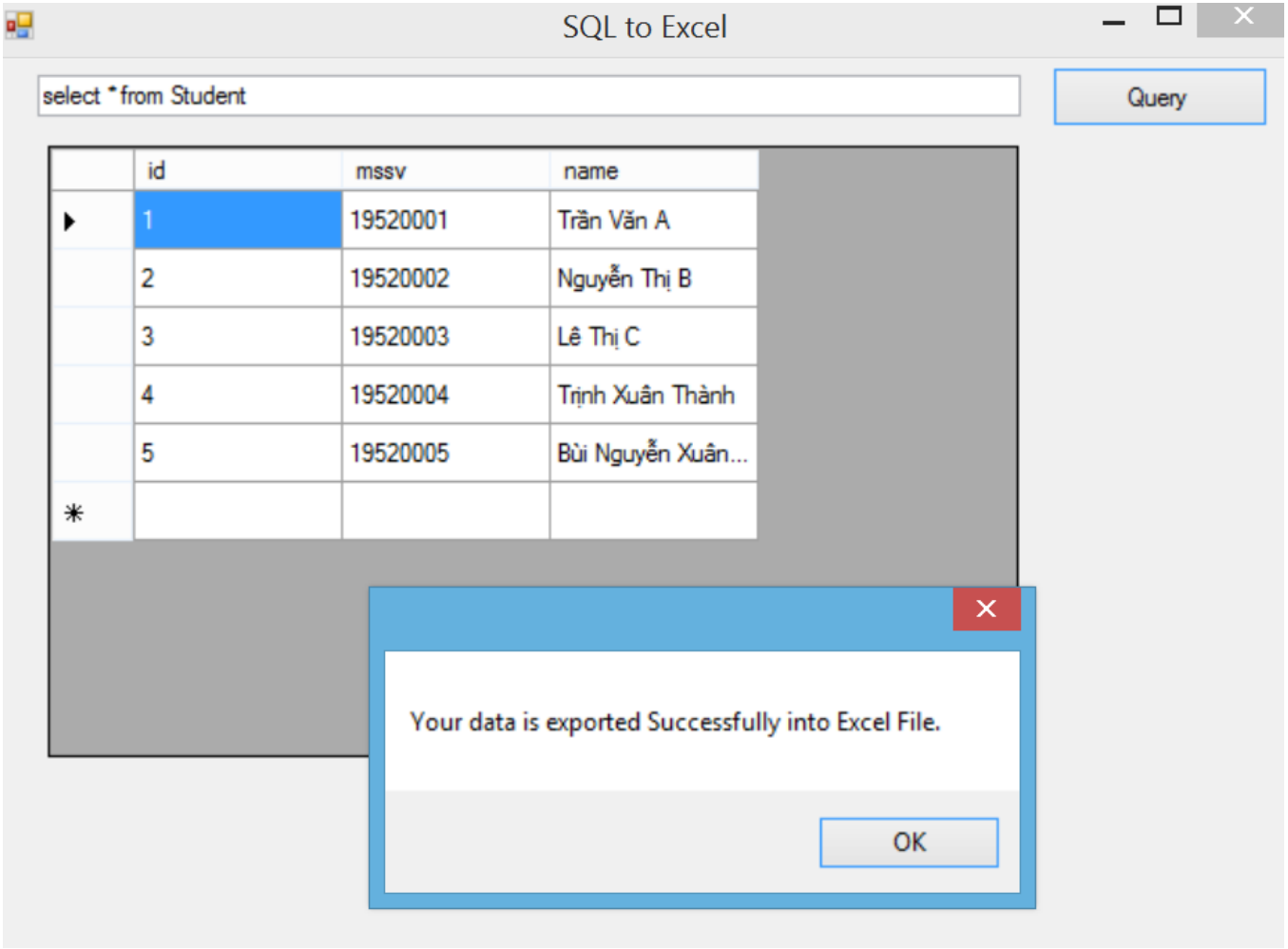
DATASET SERIALIZATION

```
OleDbCommand odbc = new
OleDbCommand(tbSQL.Text,DSN);

OleDbDataAdapter odda = new OleDbDataAdapter(odbc);

odda.Fill(ds,"sql");

TextWriter tw = new StreamWriter("..\\sql.xml");

xs.Serialize(tw, ds);

tw.Close();

DSN.Close();
```

SQL TO XML


SQL to XML
select * from company
Query
<NewDataSet>
- <companies
diffgr:id="companies1"
msdata:rowOrder="0">
<id>1</id>
<name>Wiley E
Coyote</name>
</companies>
- <companies
diffgr:id="companies2"
msdata:rowOrder="1">
<id>2</id>
<name>Acme

SQL TO EXCEL


SQL to Excel
select * from Student
Query
id
mssv
name
1
19520001
Trần Văn A
2
19520002
Nguyễn Thị B
3
19520003
Lê Thị C
4
19520004
Trịnh Xuân Thành
5
19520005
Bùi Nguyễn Xuân...
Your data is exported Successfully into Excel File.
OK

BÀI TẬP

- Cài đặt các chương trình đã minh họa trong bài giảng của chương bằng ngôn ngữ C#