Part I: Hardware Interfaces

Lesson1: Parallel Port

Lesson2: Serial Port

Lesson3: USB Devices

Part I I: Software Interfaces

Lesson4: Java GUI

Lesson5: Windows Forms

Lesson6: Java Web Services

Lesson7: C# Web Services

Hardware Interfaces

Lesson 1

Parallel Port

1. Introduction

PC parallel port can be very useful I/O channel for connecting your own circuits to PC. The PC's parallel port can be used to perform some hardware interfacing experiments. The port is very easy to use after you first understand some basic tricks. This paper tries to show those tricks in an easy to understand way. You can see the parallel port connector in the real panel of your PC. It is a 25 pin female (DB25) connector (to which printer is connected). In some PCs only one parallel port is present, but you can add more by buying and inserting ISA/PCI parallel port cards. In modern personal computers is present only the USB for peripheral devices connection. The pin outs of DB25 connector are shown in Figure 1.

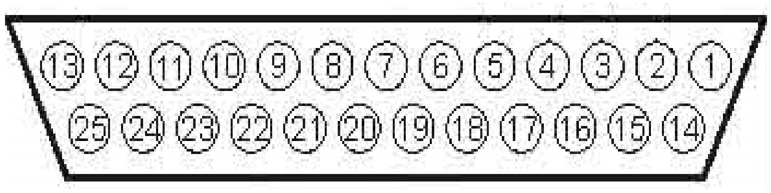


Figure 1: DB25 connector

The lines in DB25 connector are divided in to three groups, they are:

- ***Data lines (data bus)***
- ***Control lines***
- ***Status lines***

As the name refers, data is transferred over data lines, Control lines are used to control the peripheral and of course, the peripheral returns status signals back computer through Status lines. These lines are connected to Data, Control and Status registers internally. The details of parallel port signal lines are given in Table 1.

Table 1:Pin Assignments of the D-Type 25 pin Parallel Port Connector

Pin No (D-Type 25)	**SPP Signal**	**Direction In/out**	**Register**	**Hardware Inverted**
1	nStrobe	In/Out	Control	Yes
2	Data 0	Out	Data	
3	Data 1	Out	Data	
4	Data 2	Out	Data	
5	Data 3	Out	Data	
6	Data 4	Out	Data	
7	Data 5	Out	Data	
8	Data 6	Out	Data	
9	Data 7	Out	Data	
10	nAck	In	Status	
11	Busy	In	Status	Yes
12	Paper-Out / Paper-End	In	Status	
13	Select	In	Status	
14	nAuto-Linefeed	In/Out	Control	Yes
15	nError / nFault	In	Status	
16	nInitialize	In/Out	Control	
17	nSelect-Printer / nSelect-In	In/Out	Control	Yes
18 - 25	Ground	Gnd		

- http://www.beyondlogic.org/spp/parallel.htm#1

2. Parallel port registers

The Data, Control and status lines are connected to there corresponding registers inside the computer. So by manipulating these registers in program, one can easily read or write to parallel port with programming languages like C.

The registers found in standard parallel port are:

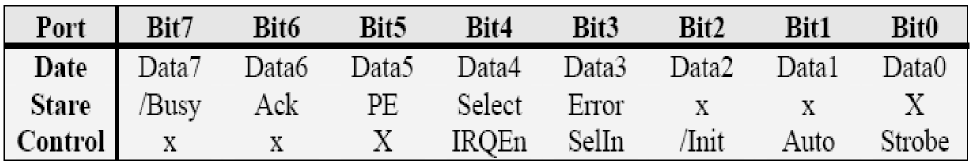
- ***Data register***
- ***Status register***
- ***Control register***

As their names specifies, Data register is connected to Data lines, Control register is connected to control lines and Status register is connected to Status lines. (Here the word connection does not mean that there is some physical connection between data/control/status lines. The registers are virtually connected to the corresponding lines.). So whatever you write to these registers, will appear in corresponding lines as voltages. And whatever you give to Parallel port as voltages can be read from these registers (with some restrictions). For example, if we write '1' to Data register, the line Data0 will be driven to +5v. Just like this, we can programmatically turn on and off any of the data lines and Control lines.

In an IBM PC, these registers are IO mapped and will have unique address. We have to find these addresses to work with parallel port. For a typical PC, the base address of LPT1 is 0x378 and of LPT2 is 0x278. The data register resides at this base address, status register at baseaddress + 1 and the control register is at baseaddress + 2. So once we have the base address, we can calculate the address of each registers in this manner. The Table 2 shows the register addresses of LPT1 and LPT2, Figure 2, shows the signals for each port.

Table 2: Port Addresses

Register	**LPT1**	**LPT2**
data registar(baseaddress + 0)	0x378	0x278
status register (baseaddress + 1)	0x379	0x279
control register (baseaddress + 2)	0x37a	0x27a



Port	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Date	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0
Stare	/Busy	Ack	PE	Select	Error	x	x	X
Control	x	x	X	IRQEn	SelIn	/Init	Auto	Strobe

Figure 2: Ports Signals

3. Programming parallel port in C under Linux

Routines for accessing I/O ports are in /usr/include/asm/io.h (or linux/include/asm-i386/io.h in the kernel source distribution). The routines there are inline macros, so it is enough to #include <asm/io.h>.

- ***Permissions:***

Before you access any ports, you must give your program permission to do so. This is done by calling the ioperm() function (declared in unistd.h, and defined in the kernel) somewhere near the start of your program (before any I/O port accesses). The syntax is ioperm(from, num, turn_on), where from is the first port number to give access to, and num the number of consecutive ports to give access to. For example, ioperm(0x300, 5, 1) would give access to ports 0x300 through 0x304 (a total of 5 ports). The last argument is a Boolean value specifying whether to give access to the program to the ports (true (1)) or to remove access (false (0)). You can call ioperm() multiple times to enable multiple non-consecutive ports.

The ioperm() call requires your program to have root privileges; thus you need to either run it as the root user. You can drop the root privileges after you have called ioperm() to enable the ports you want to use. You are not required to explicitly drop your port access privileges with ioperm(..., 0) at the end of your program; this is done automatically as the process exits.

- ***Accessing the ports:***

To input a byte (8 bits) from a port, call inb(port), it returns the byte it got. To output a byte, call outb(value, port). To input a word (16 bits) from ports x and x+1 (one byte from each to form the word, using the assembler instruction inw), call inw(x). To output a word to the two ports, use outw(value, x). If you're unsure of which port instructions (byte or word) to use, you probably want inb() and outb(); most devices are designed for byte wise port access. Note that all port access instructions take at least about a microsecond to execute.

- ***Example:***

```
#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>

#define BASEPORT 0x378 /* lp1 */

int main()
{

  /* Get access to the ports */
  ioperm(BASEPORT, 3, 1);

  /* Set the data signals (D0-7) of the port to all low (0) */
  outb(0, BASEPORT);

  /* Sleep for a while (1 s) */
  sleep(1);

  /* Read from the status port (BASE+1) and display the result */
  printf("status: %d\n", inb(BASEPORT + 1));
```

```
 /* We don't need the ports anymore */
 ioperm(BASEPORT, 3, 0);

}
```

- http://www.faqs.org/docs/Linux-mini/IO-Port-Programming.html

4. Class work

Ex1: Using the device presented in Figure 3, create an application for monitoring the parallel ports:

- Green Led – data port
- Red Led – status port
- Yellow Led – control port

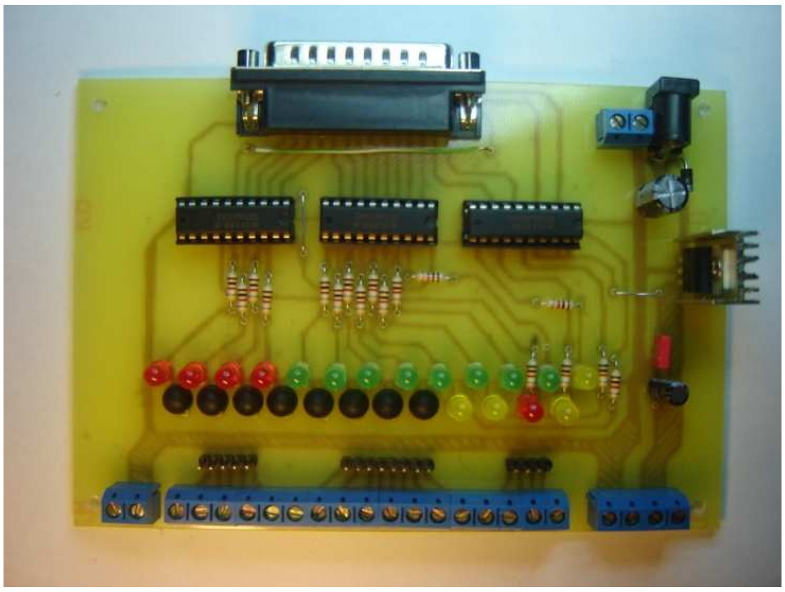


Figure 3: Parallel port interfacing device

The electrical circuit is presented in Figure 4.

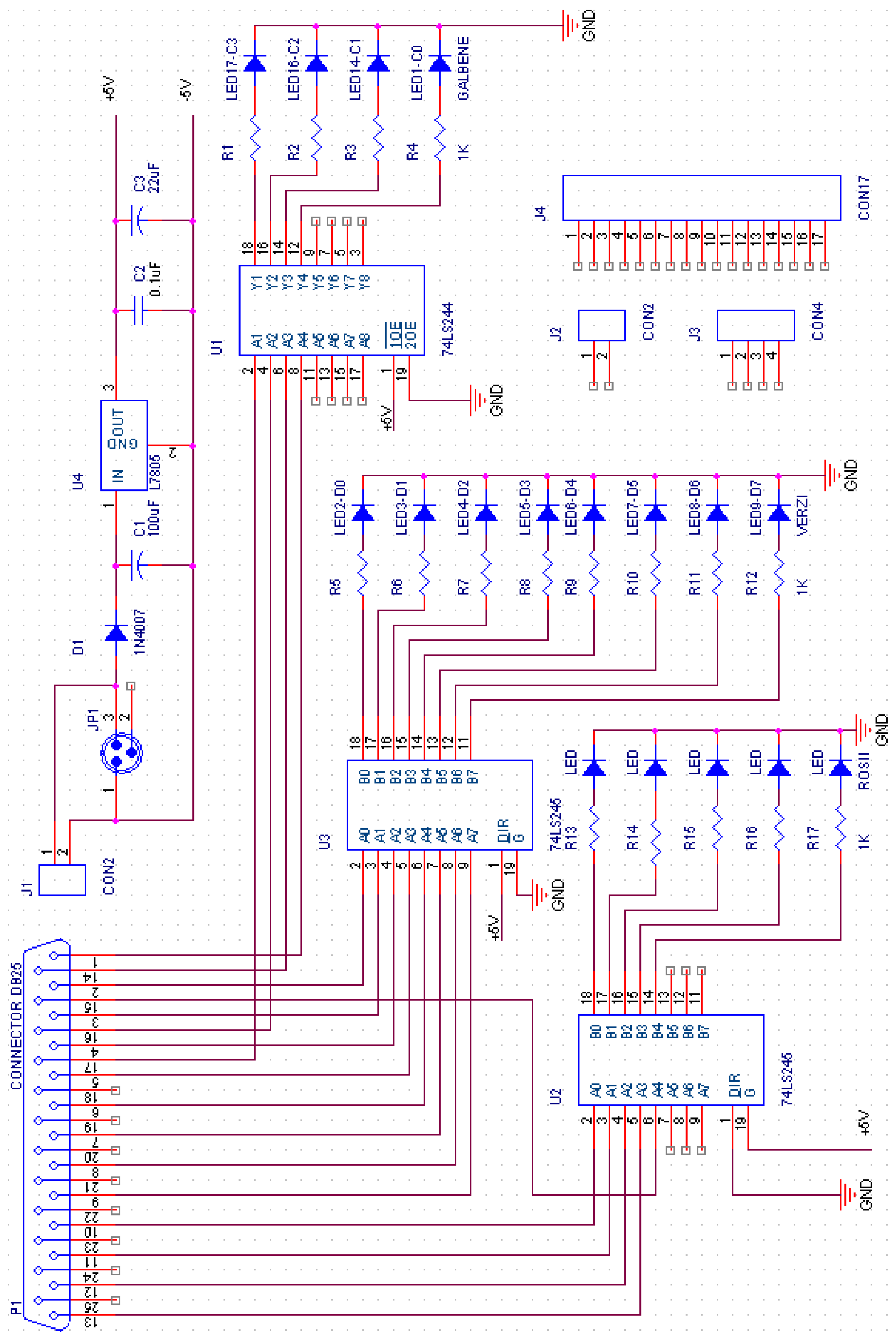


Figure 4: Electrical schema for parallel ports interfacing device

Ex2: Using the device presented in Figure 5, create an application which will command the 8 segments display.

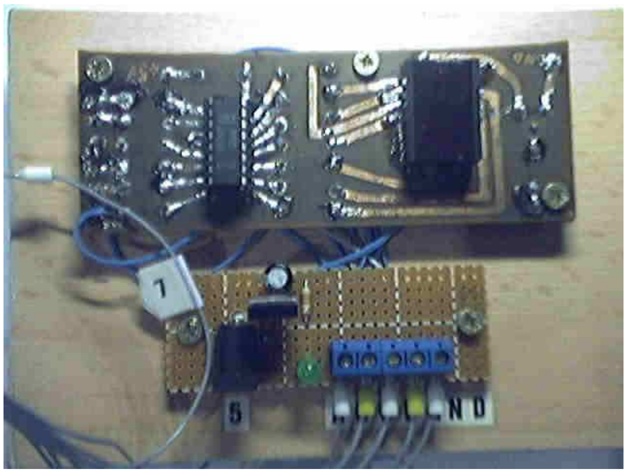


Figure 5: 8-segment display

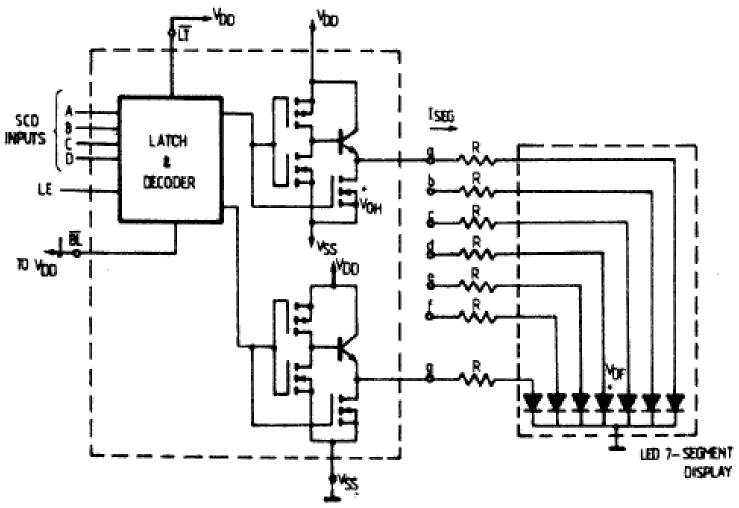
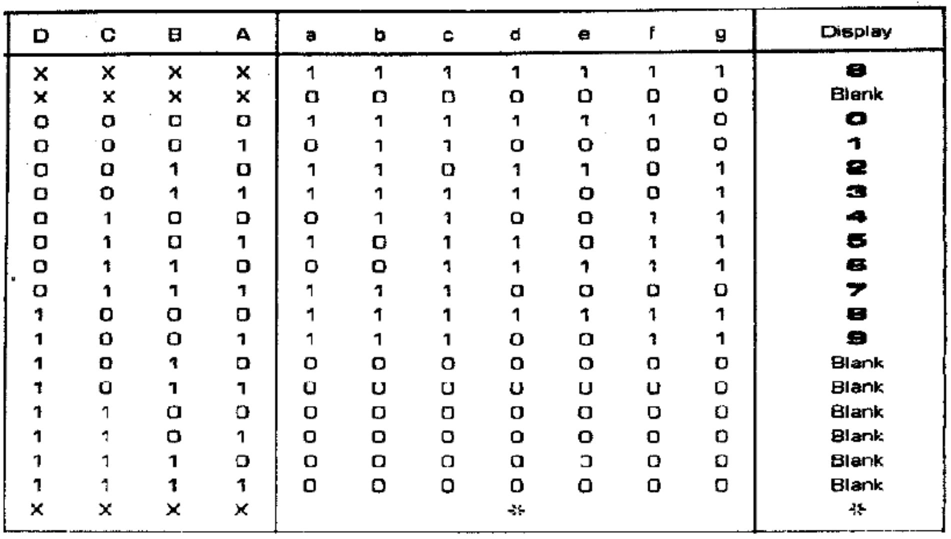


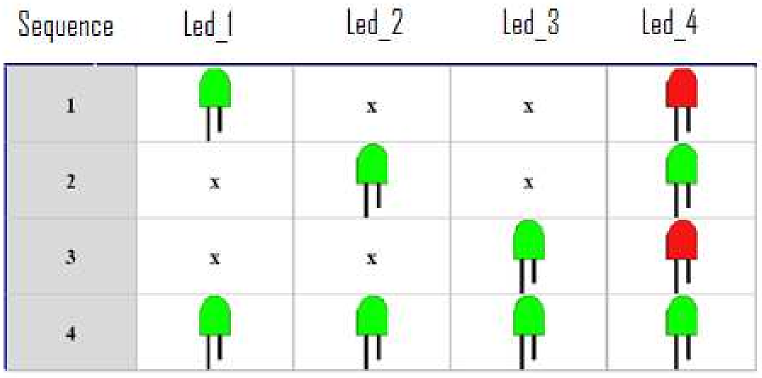
Figure 6: Electrical schema



D	C	B	A	a	b	c	d	e	f	g	Display
X	X	X	X	1	1	1	1	1	1	1	8
X	X	X	X	0	0	0	0	0	0	0	Blank
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9
1	0	1	0	0	0	0	0	0	0	0	Blank
1	0	1	1	0	0	0	0	0	0	0	Blank
1	1	0	0	0	0	0	0	0	0	0	Blank
1	1	0	1	0	0	0	0	0	0	0	Blank
1	1	1	0	0	0	0	0	0	0	0	Blank
1	1	1	1	0	0	0	0	0	0	0	Blank
X	X	X	X				*				*

Figure 7: Control table

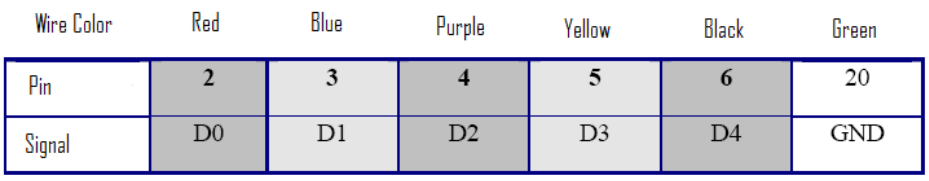
Ex3: Using the didactic device creates an application, which will turn on/off the led's like in Figure 8.



Sequence	Led_1	Led_2	Led_3	Led_4
1	🟢	x	x	🔴
2	x	🟢	x	🟢
3	x	x	🟢	🔴
4	🟢	🟢	🟢	🟢

Figure 8: Led's sequence

The signals used for the led's connection are presented in Figure 9.



Wire Color	Red	Blue	Purple	Yellow	Black	Green
Pin	2	3	4	5	6	20
Signal	D0	D1	D2	D3	D4	GND

Figure 9: Led's signals

Ex4: Using the didactic device presented in Figure 10, create an application to control barrier movements. The electrical circuit is presented in Figure 11.

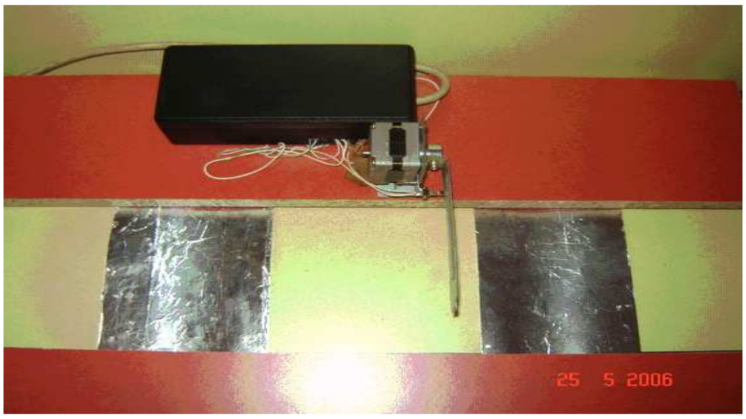


Figure 10: Barrier didactic device

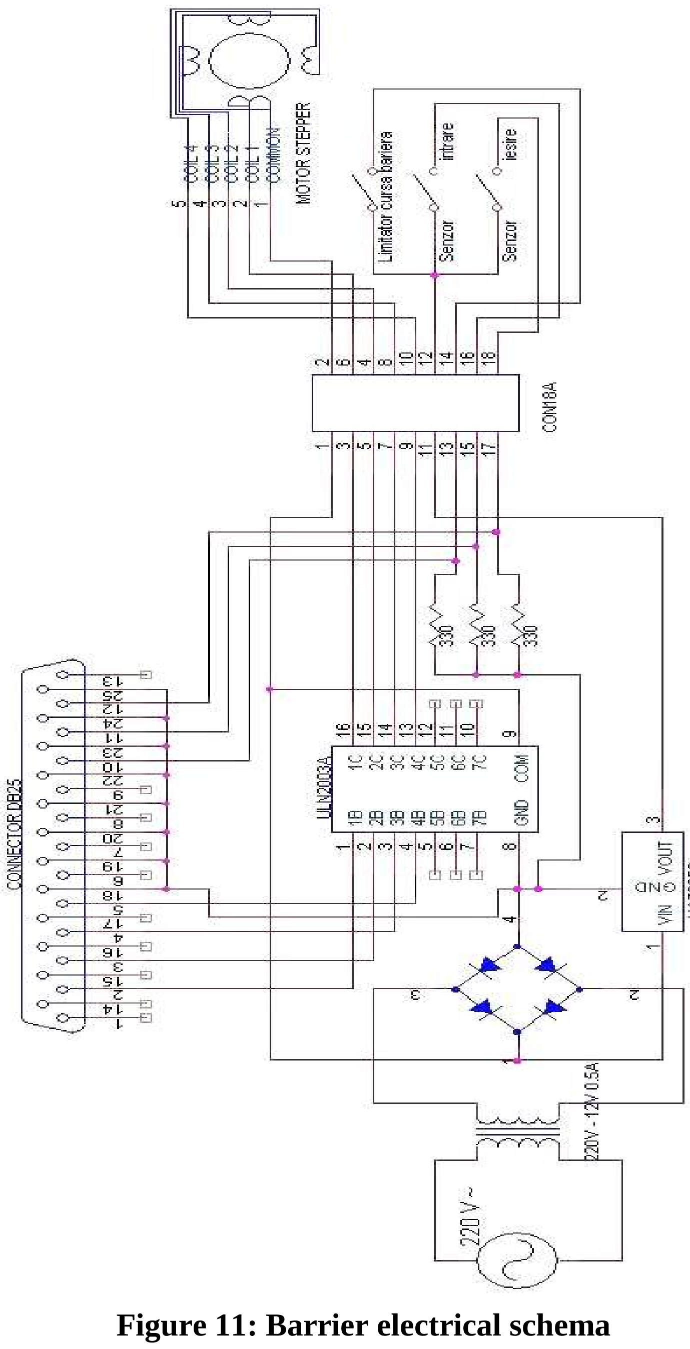


Figure 11: Barrier electrical schema

5. Home work

Ex1: Create a GUI monitoring application for parallel ports.

Lesson 2
Serial Port

1. Introduction

In computing, a serial port is a serial communication physical interface through which information transfers in or out one bit at a time (in contrast to the parallel port). Throughout most of the history of personal computers, data transfer through serial ports connected the computer to devices such as terminals and various peripherals.

In modern personal computers the serial port has largely been replaced by USB and Firewire for connections to peripheral devices. Many modern personal computers do not have a serial port since this legacy port has been superseded for most uses. Serial ports are commonly still used in applications such as industrial automation systems, scientific analysis and some industrial and consumer products. Server computers may use a serial port as a control console for diagnostics. Network equipment (such as routers and switches) often use serial console for configuration. Serial ports are still used in these areas as they are simple, cheap and their console functions are highly standardized and widespread. A serial port requires very little supporting software from the host system.

The electrical specifications of the serial port is contained in the EIA (Electronics Industry Association) RS232C standard. It states many parameters such as:

- Logic 0 - will be between +3 and +25 Volts.
- Logic 1 - will be between -3 and -25 Volts.
- The region between +3 and -3 volts is undefined.

- An open circuit voltage should never exceed 25 volts. (In Reference to GND).
- A short circuit current should not exceed 500mA. The driver should be able to handle this without damage.

The RS-232D has existed in two types: D-TYPE 25(Table 1) pin connector and D-TYPE 9(Table 2) pin connector, which are male connectors on the back of the PC. You need a female connector on your communication from Host to Guest computer.

Table 1: 25 Pin Connector on a DTE device (PC connection)

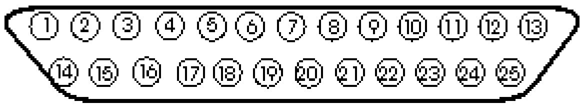
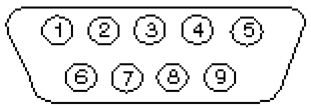
Male RS232 DB25	 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Pin Number	**Direction of signal**
1	Protective Ground
2	Transmitted Data (TD) Outgoing Data (from a DTE to a DCE)
3	Received Data (RD) Incoming Data (from a DCE to a DTE)
4	Request To Send (RTS) Outgoing flow control signal controlled by DTE
5	Clear To Send (CTS) Incoming flow control signal controlled by DCE
6	Data Set Ready (DSR) Incoming handshaking signal controlled by DCE
7	Signal Ground Common reference voltage
8	Carrier Detect (CD) Incoming signal from a modem
20	Data Terminal Ready (DTR) Outgoing handshaking signal controlled by DTE
22	Ring Indicator (RI) Incoming signal from a modem

Table 2: Pin Connector on a DTE device (PC connection)

Male RS232 DB9	
Pin Number	**Direction of signal**
1	Carrier Detect (CD) (from DCE) Incoming signal from a modem
2	Received Data (RD) Incoming Data from a DCE
3	Transmitted Data (TD) Outgoing Data to a DCE
4	Data Terminal Ready (DTR) Outgoing handshaking signal
5	Signal Ground Common reference voltage
6	Data Set Ready (DSR) Incoming handshaking signal
7	Request To Send (RTS) Outgoing flow control signal
8	Clear To Send (CTS) Incoming flow control signal
9	Ring Indicator (RI) (from DCE) Incoming signal from a modem

- http://www.beyondlogic.org/serial/serial.htm

2. Serial Port's Registers (PC's)

- ***Port Addresses(Table 3)***

Table 3: Standard Port Addresses

Name	**Address**	**IRQ**
COM 1	3F8	4
COM 2	2F8	3
COM 3	3E8	4
COM 4	2E8	3

- ***Table of Registers(Table 4)***

Table 4: Table of Registers

Base Address	**DLAB**	**Read/Write**	**Abr.**	**Register Name**
+ 0	=0	Write	-	Transmitter Holding Buffer
	=0	Read	-	Receiver Buffer
	=1	Read/Write	-	Divisor Latch Low Byte
+ 1	=0	Read/Write	IER	Interrupt Enable Register
	=1	Read/Write	-	Divisor Latch High Byte
+ 2	-	Read	IIR	Interrupt Identification Register
	-	Write	FCR	FIFO Control Register
+ 3	-	Read/Write	LCR	Line Control Register
+ 4	-	Read/Write	MCR	Modem Control Register
+ 5	-	Read	LSR	Line Status Register
+ 6	-	Read	MSR	Modem Status Register
+ 7	-	Read/Write	-	Scratch Register

- ***Used Baudrate Divisors(Table 5)***

Table 5: Table of Commonly Used Baudrate Divisors

Speed (BPS)	**Divisor (Dec)**	**Divisor Latch High Byte**	**Divisor Latch Low Byte**
50	2304	09h	00h
300	384	01h	80h
600	192	00h	C0h
2400	48	00h	30h
4800	24	00h	18h
9600	12	00h	0Ch
19200	6	00h	06h
38400	3	00h	03h
57600	2	00h	02h

115200	1	00h	01h

- ***Interrupt Enable Register (Table 6)***

Table 6: Interrupt Enable Register

Bit	**Notes**
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Enables Low Power Mode (16750)
Bit 4	Enables Sleep Mode (16750)
Bit 3	Enable Modem Status Interrupt
Bit 2	Enable Receiver Line Status Interrupt
Bit 1	Enable Transmitter Holding Register Empty Interrupt
Bit 0	Enable Received Data Available Interrupt

- ***Interrupt Identification Register (Table 7)***

Table 7: Interrupt Identification Register

Bit	**Notes**		
Bits 6 and 7	Bit 6	Bit 7	
	0	0	No FIFO
	0	1	FIFO Enabled but Unusable
	1	1	FIFO Enabled
Bit 5	64 Byte Fifo Enabled (16750 only)		
Bit 4	Reserved		
Bit 3	0	Reserved on 8250, 16450	
	1	16550 Time-out Interrupt Pending	
Bits 1	Bit	Bit	

and 2	2	1	
	0	0	Modem Status Interrupt
	0	1	Transmitter Holding Register Empty Interrupt
	1	0	Received Data Available Interrupt
	1	1	Receiver Line Status Interrupt
Bit 0	0	Interrupt Pending	
	1	No Interrupt Pending	

- ***First In / First Out Control Register (Table 8)***

Table 8: FIFO Control Register

Bit	**Notes**		
Bits 6 and 7	Bit 7	Bit 6	Interrupt Trigger Level
	0	0	1 Byte
	0	1	4 Bytes
	1	0	8 Bytes
	1	1	14 Bytes
Bit 5	Enable 64 Byte FIFO (16750 only)		
Bit 4	Reserved		
Bit 3	DMA Mode Select. Change status of RXRDY & TXRDY pins from mode 1 to mode 2.		
Bit 2	Clear Transmit FIFO		
Bit 1	Clear Receive FIFO		
Bit 0	Enable FIFO's		

- ***Line Control Register (Table 9)***

Table 9: Line Control Register

Bit 7	1	Divisor Latch Access Bit		
	0	Access to Receiver buffer, Transmitter buffer & Interrupt Enable Register		
Bit 6	Set Break Enable			
Bits 3, 4 And 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0	One Stop Bit		
	1	2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.		
Bits 0 And 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

3. Programming serial port in C under Linux

Routines for accessing I/O ports are in /usr/include/asm/io.h (or linux/include/asm-i386/io.h in the kernel source distribution). The routines there are inline macros, so it is enough to #include <asm/io.h>.

- ***Permissions***

Before you access any ports, you must give your program permission to do so. This is done by calling the ioperm() function (declared in unistd.h, and defined in the kernel) somewhere near the start of your program (before any I/O port accesses). The syntax is ioperm(from, num, turn_on), where from is the first port number to give access to, and num the number of consecutive ports to give access to. For example, ioperm(0x300, 5, 1) would give access to ports 0x300 through 0x304 (a total of 5 ports). The last argument is a Boolean value specifying whether to give access to the program to the ports (true (1)) or to remove access (false (0)). You can call ioperm() multiple times to enable multiple non-consecutive ports.

The ioperm() call requires your program to have root privileges; thus you need to either run it as the root user. You can drop the root privileges after you have called ioperm() to enable the ports you want to use. You are not required to explicitly drop your port access privileges with ioperm(..., 0) at the end of your program; this is done automatically as the process exits.

- ***Accessing the ports:***

To input a byte (8 bits) from a port, call inb(port), it returns the byte it got. To output a byte, call outb(value, port). To input a word (16 bits) from ports x and x+1 (one byte from each to form the word, using the assembler instruction inw), cal inw(x). To output a word to the two ports, use outw(value, x). If you're unsure of which port instructions (byte or word) to use, you probably want inb() and outb() --- most devices are designed for bytewise port access. Note that all port access instructions take at least about a microsecond to execute.

- ***Example***

```
#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>
#define port 0x3F8
/* Defines Serial Ports Base Address */

/* COM1 0x3F8 */
/* COM2 0x2F8 */
/* COM3 0x3E8 */
/* COM4 0x2E8 */
int main()
{
char c='a';
outb(0, port + 1); /* Turn off interrupts - Port1 */
/* port - Communication Settings */
outb(0x80, port + 3); /* SET DLAB ON */
outb(0x03, port + 0); /* Set Baud rate - Divisor Latch Low Byte */
/* Default 0x03 = 38,400 BPS */
/* 0x01 = 115,200 BPS */
/* 0x02 = 57,600 BPS */
/* 0x06 = 19,200 BPS */
/* 0x0C = 9,600 BPS */
/* 0x18 = 4,800 BPS */
/* 0x30 = 2,400 BPS */
outb(0x00, port + 1); /* Set Baud rate - Divisor Latch High Byte */
outb(0x03, port + 3); /* 8 Bits, No Parity, 1 Stop Bit */
outb(0xc7, port + 2); /* FIFO Control Register */
outb(0x0B, port + 4); /* Turn on DTR, RTS, and OUT2 */
printf("\nSample Comm's Program\n");
outb(c, port); /* Send Char to Serial Port */
}
```

- http://www.faqs.org/docs/Linux-mini/IO-Port-Programming.html

4. Class work

Ex1: Create a cable connection like in Figure 1.

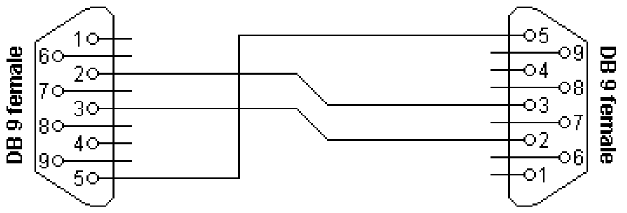


Figure 1: Simple cable connection

Ex2: Create an application that send the char ‚a’ form PC1, to PC2, and display the char on the PC2 console.

Ex3: Create an application that send a string (the string will be read from the input) form PC1, to PC2, and display the string on the PC2 console.

Ex4: Create an application that send an int, bigger than 255, form PC1, to PC2, and display the int on the PC2 console.

5. Home Work

Ex1: Create an application for a bidirectional communication, where each PC1 and PC2 can transmit and receive data.

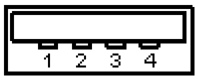
Lesson 3

USB Devices

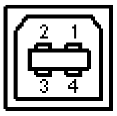
1. Introduction

USB is the most successful personal-computer interface ever. Every recent PC has USB ports that can connect to keyboards, mice, game controllers, scanners, cameras, printers, drives, and more. USB is reliable, fast, versatile, power-conserving, inexpensive, and supported by major operating systems. USB is a likely solution any time you want to use a computer to communicate with an external device. Internal devices, such as fingerprint readers, can use USB as well. The interface is suitable for mass-produced, consumer devices as well as specialized, small-volume products and one-of-a-kind projects. To be successful, an interface has to please two audiences: the users who want to use the devices and the developers who design the hardware and write the code that communicates with the devices. USB has features to please both groups.

All USB devices have an upstream connection to the host and all hosts have a downstream connection to the device. Upstream and downstream connectors are not mechanically interchangeable, thus eliminating illegal loopback connections at hubs such as a downstream port connected to a downstream port. There are commonly two types of connectors, called type A and type B which are shown below.



Type A USB Connector



Type B USB Connector

Type A plugs always face upstream. Type A sockets will typically find themselves on hosts and hubs. For example type A sockets are common on computer main boards and hubs. Type B plugs are always connected downstream and consequently type B sockets are found on devices.

The maximum length of a standard USB cable (for USB 2.0 or earlier) is 5.0 metres.

Table 1: USB 1.x/2.0 cable wiring

Pin	**Name**	**Cable color**	**Description**
1	VCC	Red	+5 V
2	D-	White	Data −
3	D+	Green	Data +
4	GND	Black	Ground

USB communication takes the form of packets. Initially, all packets are sent from the host, via the root hub and possibly more hubs, to devices. Some of those packets direct a device to send some packets in reply. All packets are made of 8-bit bytes, transmitted least-significant bit first. The first byte is a packet identifier (PID) byte. The PID is actually 4 bits; the byte consists of the 4-bit PID followed by its bitwise complement. This redundancy helps detect errors. (Note also that a PID byte contains at most four consecutive 1 bits, and thus will never need bit-stuffing, even when combined with the final 1 bit in the sync byte. However, trailing 1 bits in the PID may require bit-stuffing within the first few bits of the payload.)

Communication on the USB network can use any one of four different data transfer types:

- ***Control transfers:*** These are short data packets for device control and configuration, particularly at attach time;
- ***Bulk transfers:*** These are data packets in relatively large quantities. Devices like scanners or SCSI adapters use this transfer type;
- ***Interrupt transfers:*** These are data packets that are polled periodically. The host controller will automatically post an interrupt at a specified interval;
- ***Isochronous transfers:*** These are data streams in real time with higher requirements for bandwidth than for reliability. Audio and video devices generally use this transfer type;

Like a serial port, each USB port on a computer is assigned a unique identification number (port ID) by the USB controller. When a USB device is attached to a USB port, this unique port ID is assigned to the device and the *device descriptor* is read by the USB controller The device descriptor includes information that applies globally to the device, as well as information on the *configuration* of the device. A configuration defines the functionality and I/O behavior of a USB device. A USB device may have one or more configurations, which are described by their corresponding configuration descriptors. Each configuration has one or more *interfaces*, which can be considered as a physical communication channel. Each interface has zero or more endpoints, which can be either data providers or data consumers, or both. Interfaces are described by interface descriptors, and endpoints are described by end-point descriptors. Furthermore, a USB device might also have string descriptors to provide additional information such as vendor name, device name, or serial numbers.

- http://www.usb.org/
- http://www.beyondlogic.org/usbnutshell/usb2.htm

2. Accessing USB devices from Java applications

Most general-purpose operating systems provide support for USB devices, and it is relatively easy to develop applications in C or C++ that access such peripherals. However, the Java programming language by design provides very little support for hardware access, so writing Java applications that interact with USB devices has proved quite difficult.

The jUSB project was created by Mojo Jojo and David Brownell in June 2000. Its objective was to provide a set of free software Java APIs to access USB devices on Linux platforms. The API is distributed under the Lesser GPL (LGPL), which means that you can use it in proprietary as well as free software projects. The API provides multithreaded access to multiple physical USB devices, and supports both native and remote devices. Devices with multiple interfaces can be accessed by multiple applications (or device drivers) simultaneously, with each application (or device driver) claiming a different interface. The API supports control transfers, bulk transfers, and interrupt transfers. Currently, the API works on GNU/Linux distributions with either the Linux 2.4 kernel or a back port into 2.2.18 kernel. For example, the API works on Red Hat 7.2 and 9.0 without any patches or other upgrades.

The jUSB API includes the following packages:

- usb.core: This package is the core part of the jUSB API. It allows Java applications to access USB devices from USB hosts;
- usb.linux: This package contains a Linux implementation of a usb.core.Host object, bootstrapping support, and other classes leveraging Linux USB support. This implementation accesses the USB devices through the virtual USB device file system (usbdevfs);
- usb.windows: This package has a Windows implementation of a usb.core.Host object, bootstrapping support, and other classes leveraging Windows USB support. This implementation is still in its very early stage;
- usb.remote: This package is a remote version of the usb.core API. It includes an RMI proxy and a daemon application, which allow Java applications to access USB devices on a remote computer;

- usb.util: This package provides some useful utilities to download firmware to USB devices, dump the content of the USB system into XML, and convert a USB device with only bulk I/O into a socket;
- usb.devices: This optional package collects Java code to access a variety of USB devices with the jUSB API, including Kodak digital cameras and Rio 500 MP3 Players. These APIs were specially written to simplify the process of accessing the designated USB devices and cannot be used to access other devices. The APIs were built upon the usb.core APIs, and they will work on any operating system where jUSB is supported;
- usb.view: This optional package provides a simple USB tree browser based on Swing;

Although the implementation of the usb.core. Host object varies from operating system to operating system, a Java programmer needs to understand only the usb.core package to start developing applications with the jUSB APIs. Table 2 outlines the interfaces and classes from usb.core with which a Java programmer should be familiar.

Table 2: Interfaces and classes in jUSB

Interface	**Description**
Bus	Connects a set of USB devices to a Host
Host	Represents a USB controller with one or more Buses
Class	**Description**
Configuration	Provides access to a USB configuration supported by a device and to the interfaces associated with that configuration
Descriptor	Base class for entities with USB typed descriptors
Device	Provides access to a USB device

DeviceDescriptor	Provides access to a USB device descriptor
EndPoint	Provides access to a USB end-point descriptor, structuring device data input or output in a given device configuration
HostFactory	Contains bootstrapping methods
Hub	Provides access to a USB hub descriptor and some hub operations
Interface	Describes sets of endpoints, and is associated with a particular device configuration
PortIdentifier	Provides stable string identifiers for USB devices, appropriate for use in operations and troubleshooting

The normal procedure to access a USB device with the jUSB API is as follows:

- Bootstrap by getting the USB Host from the HostFactory;
- Access the USB Bus from the Host, then access the USB root hub (which is a USB Device) from the Bus;
- Obtain the number of USB ports available on the hub, and traverse through all the ports to find the appropriate Device;
- Access the USB Device that is attached to a particular port. A Device can be accessed directly from the Host with its PortIdentifier, or can be found by traversing the USB Bus starting from the root hub;
- Interact with the Device directly with ControlMessage, or claim an Interface from the current Configuration of the Device and perform I/O with the Endpoint available on the Interface;

The next example illustrates how to obtain the content of a USB system with the jUSB API. The program as written simply looks at the root hub for available USB devices, but it would be easy to improve it to traverse the whole USB tree.

```
import usb.core.*;

public class ListUSB
{
   public static void main(String[] args)
   {
      try
      {
         // Bootstrap by getting the USB Host from the HostFactory.
         Host   host = HostFactory.getHost();

         // Obtain a list of the USB buses available on the Host.
         Bus[]  bus  = host.getBusses();
         int    total_bus = bus.length;

         // Traverse through all the USB buses.
         for (int i=0; i<total_bus; i++)
         {
            // Access the root hub on the USB bus and obtain the
            // number of USB ports available on the root hub.
            Device root = bus[i].getRootHub();
            int total_port = root.getNumPorts();

            // Traverse through all the USB ports available on the
            // root hub. It should be mentioned that the numbering
            // starts from 1, not 0.
            for (int j=1; j<=total_port; j++)
            {
               // Obtain the Device connected to the port.
               Device device = root.getChild(j);
               if (device != null)
               {
                  // USB device available, do something here.
                          }
            }
         }
      } catch (Exception e)
      {
         System.out.println(e.getMessage());
      }
   }
```

The next example illustrates how to perform bulk I/O with Interface and EndPoint, assuming that the application has successfully located the Device.

```
if (device != null)
  {
     // Obtain the current Configuration of the device and the number of
     // Interfaces available under the current Configuration.
     Configuration config = device.getConfiguration();
     int total_interface = config.getNumInterfaces();

     // Traverse through the Interfaces
     for (int k=0; k<total_interface; k++)
     {
        // Access the currently Interface and obtain the number of
        // endpoints available on the Interface.
        Interface itf = config.getInterface(k, 0);
        int total_ep  = itf.getNumEndpoints();

       // Traverse through all the endpoints.
       for (int l=0; l<total_ep; l++)
       {
           // Access the endpoint, and obtain its I/O type.
           Endpoint ep = itf.getEndpoint(l);
           String io_type = ep.getType();
           boolean input  = ep.isInput();

           // If the endpoint is an input endpoint, obtain its
           // InputStream and read in data.
           if (input)
           {
              InputStream in;
              in = ep.getInputStream();
              // Read in data here
              in.close();
           }
           // If the Endpoint is and output Endpoint, obtain its
           // OutputStream and write out data.
           else
```

```
          {
            OutputStream out;
            out = ep.getOutputStream();
            // Write out data here.
            out.close();
          }
        }
      }
    }
```

The release of the API, version 0.4.4, was made available on February 14, 2001. Only some minor progress has been reported since that time, probably due to the success of the IBM group in becoming a candidate extended standard of the Java language. However, several third-party applications have been developed based on jUSB, including the JPhoto project (an application using jUSB to connect to digital cameras) and the jSyncManager project (an application using jUSB to synchronize with a Palm OS-based PDA). The objective of the project was to develop a USB interface for the Java platform that will allow full access to the USB system for any Java application or middleware component. The JSR-80 API provides full support for all four transfer types defined by the USB specification. Currently, the Linux implementation of the API works on GNU/Linux distributions with 2.4 kernel support, such as Red Hat 7.2 and 9.0.

The JSR-80 project includes three packages: javax-usb (the javax.usb API), javax-usb-ri (the common part of the OS-independent reference implementation), and javax-usb-ri-linux (the reference implementation for the Linux platform, which connects the common reference implementation to the Linux USB stack). All three parts are required to form a complete functioning java.usb API on the Linux platform.

Although the OS-dependent implementation of the JSR-80 APIs varies from operating system to operating system, a Java programmer needs to understand only the javax.usb package to start developing applications. Table 3 lists the interfaces and classes in javax.usb with which a Java programmer should be familiar.

Table 3: Interfaces and classes in the JSR-80 APIs

Interface	**Description**
UsbConfiguration	Represents a configuration of a USB device
UsbConfigurationDescriptor	Interface for a USB configuration descriptor
UsbDevice	Interface for a USB device
UsbDeviceDescriptor	Interface for a USB device descriptor
UsbEndpoint	Interface for a USB endpoint
UsbEndpointDescriptor	Interface for a USB endpoint descriptor
UsbHub	Interface for a USB hub
UsbInterface	Interface for a USB interface
UsbInterfaceDescriptor	Interface for a USB interface descriptor
UsbPipe	Interface for a USB pipe
UsbPort	Interface for a USB port
UsbServices	Interface for a javax.usb implementation
Class	**Description**
UsbHostManager	Entry point for javax.usb

The normal procedure for accessing a USB device with the JSR-80 API is as follows:

- Bootstrap by getting the appropriate UsbServices from the UsbHostManager;
- Access the root hub through the UsbServices. The root hub is considered as a UsbHub in the application;
- Obtain a list of the UsbDevices that are connected to the root hub. Traverse through all the lower-level hubs to find the appropriate UsbDevice;
- Interact with the UsbDevice directly with a control message (UsbControlIrp), or claim a UsbInterface from the appropriate UsbConfiguration of the UsbDevice and perform I/O with the UsbEndpoint available on the UsbInterface;
- If a UsbEndpoint is used to perform I/O, open the UsbPipe associated with it. Both upstream data (from the USB device to the host computer) and downstream data (from the host computer to the USB device) can be submitted either synchronously or asynchronously through the UsbPipe;
- Close the UsbPipe and release the appropriate UsbInterface when the application no longer needs access to the UsbDevice;

The next example obtain the content of the USB system with the JSR-80 API. The program recursively traverses through all the USB hubs on the USB system and locates all the USB devices connected to the host computer.

```
import javax.usb.*;
import java.util.List;
```

```
public class TraverseUSB
{
        public static void main(String argv[])
        {
          try
          {
              // Access the system USB services, and access to the root
              // hub. Then traverse through the root hub.
              UsbServices services = UsbHostManager.getUsbServices();
              UsbHub rootHub = services.getRootUsbHub();
              traverse(rootHub);
          } catch (Exception e) {}
        }

        public static void traverse(UsbDevice device)
        {
          if (device.isUsbHub())
          {
             // This is a USB Hub, traverse through the hub.
             List attachedDevices =
                 ((UsbHub) device).getAttachedUsbDevices();
             for (int i=0; i<attachedDevices.size(); i++)
             {
               traverse((UsbDevice) attachedDevices.get(i));
             }
          }
          else
          {
             // This is a USB function, not a hub.
             // Do something.
          }
        }
}
```

The next example illustrates how to perform I/O with Interface and EndPoint, assuming that the application has successfully located a Device. This code snippet can also be modified to perform I/O of all four data transfer types.

```
public static void testIO(UsbDevice device)
{
   try
   {
      // Access to the active configuration of the USB device, obtain
      // all the interfaces available in that configuration.
      UsbConfiguration config = device.getActiveUsbConfiguration();
      List totalInterfaces = config.getUsbInterfaces();

      // Traverse through all the interfaces, and access the endpoints
      // available to that interface for I/O.
      for (int i=0; i<totalInterfaces.size(); i++)
      {
         UsbInterface interf = (UsbInterface) totalInterfaces.get(i);
         interf.claim();
         List totalEndpoints = interf.getUsbEndpoints();
         for (int j=0; j<totalEndpoints.size(); j++)
         {
            // Access the particular endpoint, determine the direction
            // of its data flow, and type of data transfer, and open the
            // data pipe for I/O.
            UsbEndpoint ep = (UsbEndpoint) totalEndpoints.get(i);
            int direction = ep.getDirection();
            int type = ep.getType();
            UsbPipe pipe = ep.getUsbPipe();
            pipe.open();
            // Perform I/O through the USB pipe here.
            pipe.close();
         }
         interf.release();
      }
   } catch (Exception e) {}
}
```

Both the jUSB API and the JSR-80 API provide Java applications with the capability to access USB devices from a machine running the Linux operating system. The JSR-80 API provides more functionality than the jUSB API, and has the potential of becoming an extended standard of the Java language.

- http://www.ibm.com/developerworks/library/j-usb.html

3. Class work

Ex1: Test the examples presented in chapter 2.

Ex2: Create a simple USB LED Light (Figure 1). Turn the led ON, wait 2 seconds and turn the led OFF.

- male USB plug;
- 22 ohm resistor;
- LED;

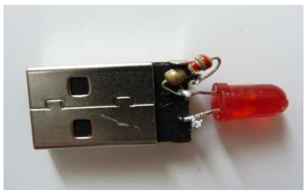


Figure 1: USB Led

4. Home Work

Ex1: Make a Universal Serial Bus presentation: History; Overview; Device classes; USB mass-storage; Human-interface devices (HIDs); Signaling; Data packets; Handshake packets; Token packets; Connector properties; Protocol analyzers; Connector types; Comparisons with other device connection technologies;

Lesson 4

Java GUI

1. Introduction

This chapter gives you a short introduction in Java Foundation Classes (JFC) and Swing.

JFC encompass a group of features for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications.

The Swing API is only one of five libraries that make up the JFC. The Java Foundation Classes also consist of the Abstract Window Toolkit (AWT), the Accessibility API, the 2D API, and enhanced support for drag-and-drop capabilities.

- AWT - basic GUI Toolkit shipped with all versions of JDK. Swing does not reuse any AWT components but does build off some lightweight component facilities;
- Accessibility - enables assistive technologies, such as screen readers and Braille displays, to get information from the user interface. All Swing Components support accessibility;
- 2D API - Various painting styles, drawing styles, fonts and colours. Not Part of SWING;
- Drag and Drop - Click and hold of GUI objects, moving windows or objects *etc*. Not Part of SWING ;

Swing , which provides enhanced versions of all AWT components and augments them with capabilities like pluggable look and feel for custom interfaces and accessibility support to help users with special needs. In the early days of Java, Sun introduced AWT as a platform-independent user interface builder that addressed the dilemna of producing code for specific operating environments. The theory suggested that a developer could design complex user interfaces on a Sun Solaris platform and easily move them to Microsoft Windows or the Apple Macintosh without having to recompile source code. In reality, AWT failed to meet this promise because the GUIs of these platforms are so different from one another.

In June 1998, Sun presented a solution to the limitations of AWT, including improved and consistent components as well as a much more solid foundation. This new design, based on the Model-View-Controller (MVC) concept, originated in the Smalltalk world and was later exploited by Erich Gamma et al in their book *Design Patterns* (Addison-Wesley, 1995). This technology, known as Swing, is one part of a much larger effort, known as Java Foundation Classes (JFC), that promises to pave a standard road into the future.

The MVC architecture offers some distinct advantages for AWT, including the capability to replace the data model with one that is custom designed for an apllication, or to redesign the user interface for individual components as required. This second point is especially important since it also accomodates nontraditional user interfaces such as speech or braille output devices for users with special needs. This capability is not possible with AWT components.

MVC breaks GUI components into 3 elements:

- ***Model***

  - *State* data for each component;
  - Different for each component;

E.g. Scrollbar - Max and minimum Values, Current Position, width of *thumb* position relative to values

E.g. Menu - Simple List of items.

  - Model data is *always* independent of visual representation;

- ***View***

  - How component is painted on the screen;
  - Can vary between platforms/look and feel;

- ***Controller***

  - dictates how component interacts with events ;
  - Many forms of events - mouse clicks, keyboard, focus, repaint *etc*.;

The three elements interact as follows (Figure 1):

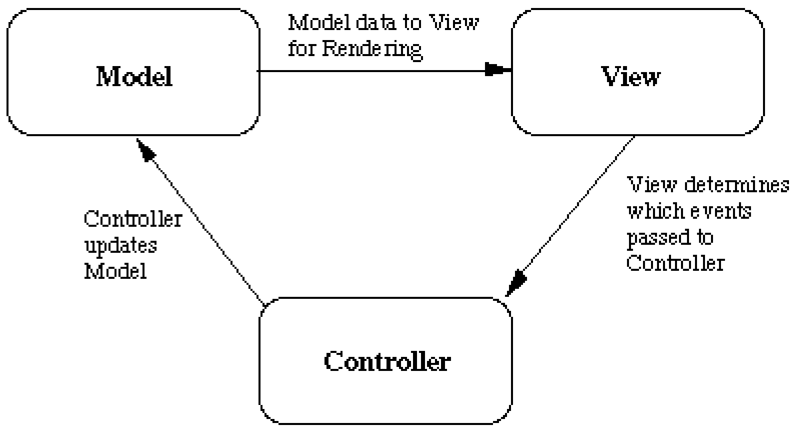

Model
View
Controller
Model data to View for Rendering
View determines which events passed to Controller
Controller updates Model

Figure 12: The MVC Communication Model

Swing actually makes use of a simplified variant of the MVC design called the *model-delegate* .This design combines the view and the controller object into a single element that draws the component to the screen and handles GUI events known as the *UI delegate* . Bundling graphics capabilities and event handling is somewhat easy in Java, since much of the event handling is taken care of in AWT. As you might expect, the communication between the model and the UI delegate then becomes a two-way street, as shown in Figure 2.

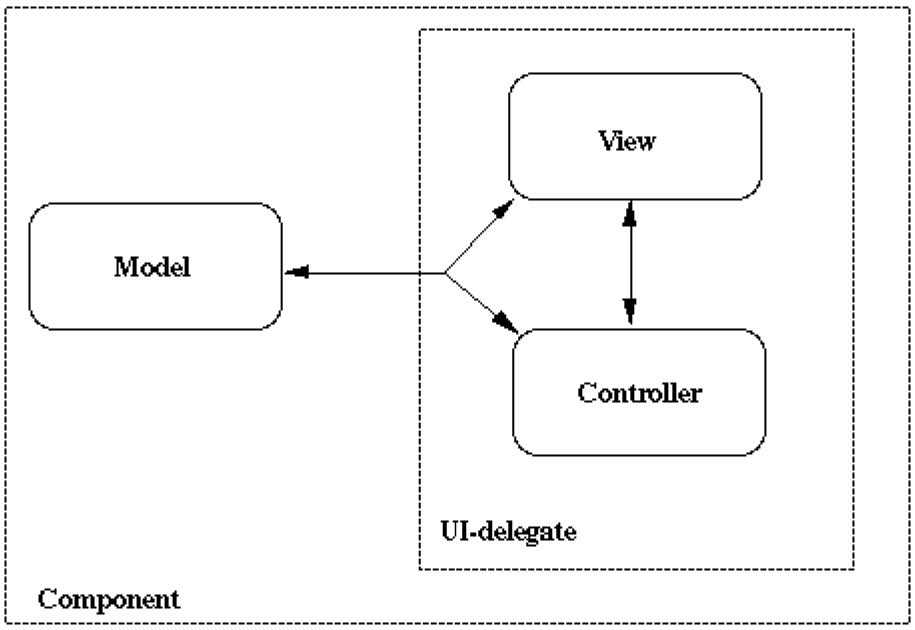


Figure 13: MVC design -> model-delegate

So each Swing component contains a model and a UI delegate. The model is responsible for maintaining information about the component's state. The UI delegate is responsible for maintaining information about how to draw the component on the screen. In addition, the UI delegate (in conjunction with AWT) reacts to various events that propagate through the component.

Architecturally, the Swing class library is similar to AWT. This is a great benefit to those migrating from the older technology to the new. Most Swing component APIs are similar and often the same as their AWT counterparts, so developers can quickly grasp Swing's basics and port older applications with relative ease. Figure 3 shows the user interface class hierarchy supported by Swing. Note that with the addition of a "J" prefix, the class names closely resemble those currently offered in AWT, though Swing offers user interface classes not found in AWT.

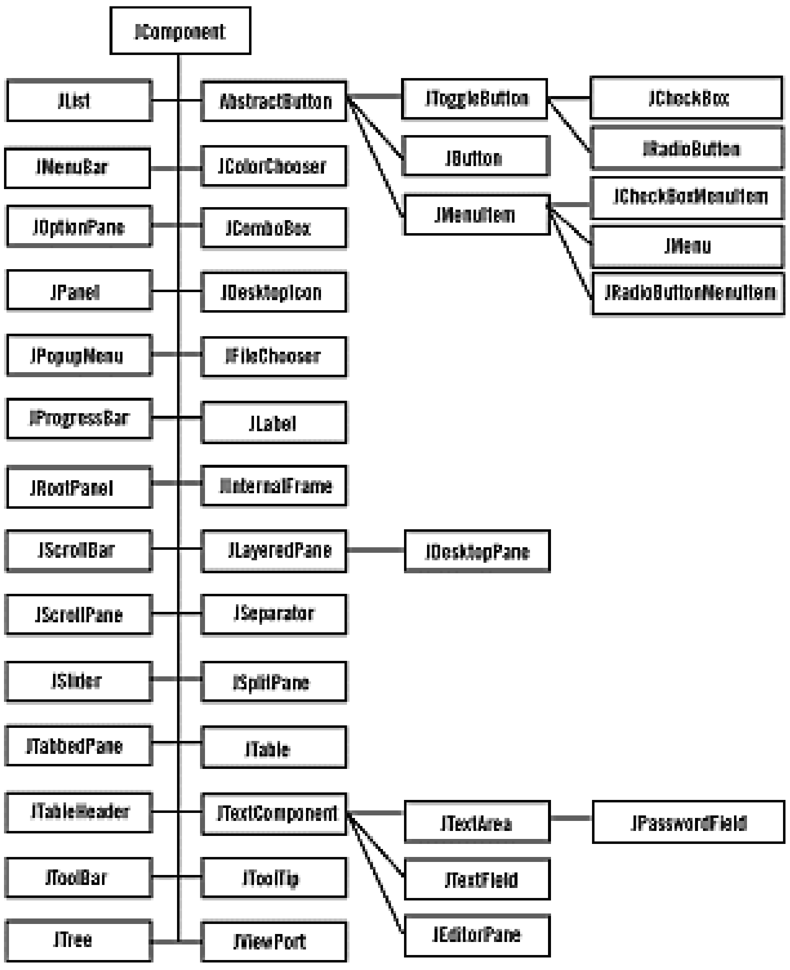


Figure 14: The Swing User Interface Classes

Figure 3 shows that all Swing components extend the JComponent class. Since JComponent extends the AWT container class, this implies that all

Swing user interface components are containers. This offers some avenues for developers to create interesting applications by building up components inside components. For example, inserting a checkbox or graphic into a list element is a relatively simple operation with Swing, but is impossible with AWT. Swing's capabilities generally start where AWT leaves off, letting developers create more advanced applications faster than they could before.

2. Using Swing Components

This chapter gives you the background information you need to use Swing components.

The Swing package defines two types of components:

- top-level containers (JFrame, JApplet, JWindow, JDialog) ;
- lightweight components (*everything-else*, such as JButton, JPanel, and JMenu) ;

- ***JFrame***

A Frame (Figure 4) is a top-level window with a title and a border. Here is a picture of the extremely plain window created by the FrameDemo demonstration application:

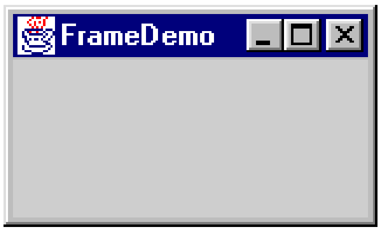


Figure 15: JFrame

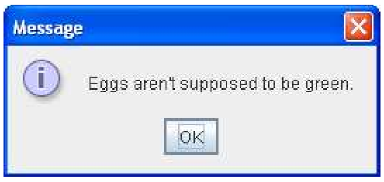
The following FrameDemo code shows how to create and set up a frame:

```
//1. Create the frame.
JFrame frame = new JFrame("FrameDemo");

//2. Optional: What happens when the frame closes?
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//3. Create components and put them in the frame.
//...create emptyLabel...
frame.getContentPane().add(emptyLabel, BorderLayout.CENTER);

//4. Size the frame.
frame.pack();

//5. Show it.
frame.setVisible(true);
```

- ***JDialog***

A Dialog window is an independent subwindow meant to carry temporary notice apart from the main Swing Application Window. Most Dialogs present an error message or warning to a user, but Dialogs can present images, directory trees, or just about anything compatible with the main Swing Application that manages them.

For example, here is an informational dialog (Figure 5):


Message

Eggs aren't supposed to be green.

Figure 16: Jdialog

Swing uses the *JOptionPane* class to provide predefined methods for each type of dialog.

Here is the code that creates and shows it:

```
JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be green.");
```

- ***JPanel***

JPanel is the most commonly used content pane. An instance of the pane is created and then added to a frame. The *add(widgetName)* method allows widgets (ie GUI controls) to be added to the pane. The way they are added is controlled by the current layout manager. JPanel defaults to FlowLayout. All other content panes default to BorderLayout.

The following is a simple template that creates a JFrame container class using inheritance. The created subclass then adds a JPanel. This custom class will form the basis of many of our GUI examples:

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;
```

```
public class Frame1 extends JFrame

{

          JPanel pane=new JPanel();

           Frame1() // the frame constructor method

          {

                    super("My Simple Frame");

                    setBounds(100,100,300,100);
                    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                    Container con=this.getContentPane(); // inherit main
frame

                    con.add(pane); // add the panel to frame

                    // customize panel here

                    // pane.add(someWidget);

                    setVisible(true); // display this frame

          }

           public static void main(String args[]) {

                    new Frame1();

          }

}
```

- http://java.sun.com/docs/books/tutorial/uiswing/components/
- http://java.sun.com/j2se/1.5.0/docs/api/

3. Class work

Ex1: Create a window with a title and some text in it.

Package	**Class**	**Method**	**Description**
javax.swing	JFrame	Constructor: JFrame(String title)	Constructs a frame (= a window with a border) with a title.
		getContentPane()	Returns a Component in which content can be placed.
		pack()	Resizes the frame to the preferred size of all its components.
		show()	Shows the frame on the screen and brings it to the front.
	JLabel	Constructor: JLabel(String text)	Constructs a label with initial text in it.

Ex2: Create a window with a button in it, and a counter which counts the number of times the user has clicked the button.

Package	**Class**	**Method**	**Description**

javax.swing	JPanel	setLayout(LayoutManager)	Sets a layout for the component. In this example the borderlayout is used.
		add(Component, int location)	This method can be used if a component has a BorderLayout. the location specifies the location: east, west, north, south or center.
	JButton	Constructor: JButton(String)	Creates a button with initial text in it
		addActionListener(ActionListener)	specifies a class which listens to events generated by clicks on this button
	JLabel	setText(String text)	Changes the text of the label.
	JFrame	setDefaultCloseOperation(int)	Specifies what happens if the close-button of the frame is pressed. Possible close operations are: EXIT_ON_CLOSE;HIDE_ON_CLOSE; DISPOSE_ON_CLOSE
java.awt	Border Layout	Constructor: BorderLayout()	A LayoutManager. This layout manager divides the panel into five subareas: east, west, north, south and center.
java.awt.event	ActionListener	actionPerformed(ActionEvent)	This method is called by the component which generates the events (in this case: clickButton).
	ActionEvent	getSource()	Returns the component which generates the

			events (in this case: clickButton)

Ex3: Create a window with an editable textfield, a button and a label in it. Whenever the button is pressed, the text from the textfield is copied to the label.

Package	**Class**	**Method**	**Description**
javax.swing	JTextField	Constructor: JTextField(int width)	Constructs an editable textfield which can hold a certain amount of characters.
		String getText()	Returns a String containing the text of the textfield.
		setText(String text)	Replaces the text of the textfield by the new String.
	JFrame	setSize(Dimension)	Sets the size of the frame in pixels.
java.awt	Dimension	Constructor: Dimension(int width, int height)	Creates a Dimension object with a certain width and height in pixels.
java.awt.event	ActionListener		
	ActionEvent		

EX4: Create a Frame with a menu in it. In the frame, a panel is placed which has two text fields and a label. The user can put numbers in the textfield. If he presses one of the menu items of the frame, the frame will ask the panel to perform the desired calculation.

Package	**Class**	**Method**	**Description**
javax.swing	JMenuBar	Constructor: JMenuBar()	Creates a menu bar.
		add(Menu)	Adds a menu to the menu bar.
		int getMenuCount()	Returns an integer which is the number of menus of the menu bar.
		Menu getMenu(int n)	Returns the *n*th menu of the menu bar
	JMenu	Constructor: JMenu(String title)	Creates a menu with a certain title.
		add(MenuItem)	Adds a menu item to the menu.
		addSeparator()	Adds a separator to the menu.
		int getItemCount()	Returns an integer which is the number of menu items of the menu (a separator is a menu item too).
		MenuItem getItem(int n)	Returns the nth menu-item.
	JMenuItem	Constructor: JMenuItem(String title)	Creates a menu item with a certain title.
		addActionListener(ActionListener)	Adds an ActionListener which listens to ActionEvents generated by the menu item.

	JFrame	setJMenuBar(JMenuBar)	Sets the MenuBar for the JFrame
java.lang	System	exit(int status)	Exits the program.
	Integer	static void parseInt(String n)	Tries to parse a string value (n) to an integer value. Throws a NumberFormatException if the String cannot be parsed.
java.awt	GridLayout	Constructor: GridLayout(int rows, int columns)	Creates a GridLayout with a specified number of rows and columns.

4. Home Work

Ex1: Create a simple calculator (**Figure 6**), according to the Model-View-Controller (MVC) pattern. The idea is to separate the application into: **UI** (The **View** and the **Controller**) and **Model.**

- **UI** will be implemented as a subclass of JFrame (View - creates the display, interacting with the Model as necessary and the Controller - responds to user requests, interacting with both the View and Controller as necessary).
- **Model** - where the actual calculations take place.

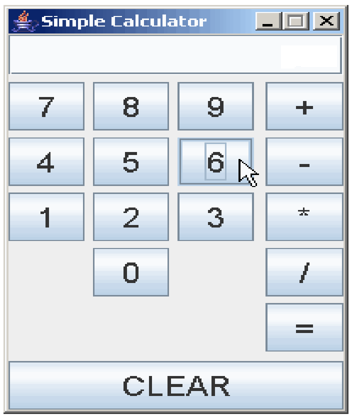


Figure 17: Simple Calculator

Ex2: Create a **X-0** game (**Figure** 7), according to the Model-View-Controller (MVC)pattern.

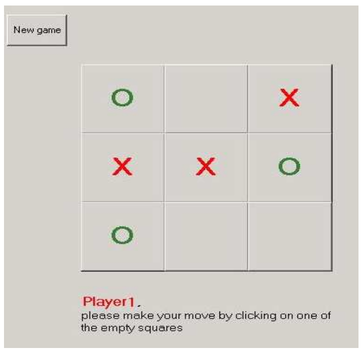

New game
O
X
X
X
O
O
Player1,
please make your move by clicking on one of the empty squares

Figure 18: X-0 game

Lesson 5

Windows Forms

1. Introduction

In this paper, we will take a quick look at the following aspects of Windows Forms programming:

- The Form class;
- Controls: how each control is a distinct class, and how controls are added to a form;

The Form - is the most fundamental object used in applications. By itself, a form does nothing. Its main role is to host other objects that the user uses to interact with the computer.

There are various ways you can get a form to your application:

- If you create a Windows Forms Application, it creates a starting form for you;
- After starting an empty project or a Windows Forms Application, you can add a form to it. To do this, on the main menu, you can click Project -> Add New Item... Select Windows Form. Give it a name and click OK;

- http://www.functionx.com/vcsharp/form/Lesson01.htm

2. Windows controls

- ***Button***

A button is a control that establishes a specific state, typically some form of on or off. Buttons are used to perform immediate actions in an interface, define the behavior for a specific feature, or turn a setting on or off. More generally, the various types of buttons are as follows:

- A *push button* - is a button that performs some immediate action, such as displaying or deactivating a dialog, or modifying the values in the window. In Windows Forms, the Button class represents a push button;
- A *check box button* - allows a user to turn a specific option on or off, such as whether a file should be saved as read-only or not. In .NET, the CheckBox class can represent either a check box button or a *toggle button*. A toggle button appears as a normal button, but preserves an up or down state to represent a checked or unchecked mode, respectively;
- A *radio button* - sometimes called an *option button*, is used to select from a set of mutually exclusive options. When one of a group of radio buttons is selected, the other radio buttons in the group are automatically deselected. Radio buttons can be displayed normally or as toggle buttons. Windows Forms provides the RadioButton class for the creation of these objects. All radio buttons in the same container are automatically part of the same group. Use container classes such as GroupBox and Panel to support multiple groups of radio buttons on your forms;

Functions:

- *CheckDlgButton* - Changes the check state of a button control;
- *CheckRadioButton* - Adds a check mark to (checks) a specified radio button in a group and removes a check mark from (clears) all other radio buttons in the group;
- *IsDlgButtonChecked* - Determines whether a button control is checked or whether a three-state button control is checked, unchecked, or indeterminate;

Macros:

- *Button_Enable* - Enables or disables a button;
- *Button_GetCheck* - Gets the check state of a radio button or check box. You can use this macro or send the BM_GETCHECK message explicitly;
- *Button_GetIdealSize* - Gets the size of the button that best fits the text and image, if an image list is present. You can use this macro or send the BCM_GETIDEALSIZE message explicitly;
- *Button_GetImageList* - Gets the BUTTON_IMAGELIST structure that describes the image list that is set for a button control. You can use this macro or send the BCM_GETIMAGELIST message explicitly;
- *Button_GetNote* - Gets the text of the note associated with a command link button. You can use this macro or send the BCM_GETNOTE message explicitly;
- *Button_GetNoteLength* - Gets the length of the note text that may be displayed in the description for a command link. Use this macro or send the BCM_GETNOTELENGTH message explicitly;

- *Button_GetSplitInfo* - Gets information for a specified split button control. Use this macro or send the BCM_GETSPLITINFO message explicitly;
- *Button_GetState* - Gets the check state of a radio button or check box. You can use this macro or send the BM_GETSTATE message explicitly;
- *Button_GetText* - Gets the text of a button;
- *Button_GetTextLength* - Gets the number of characters in the text of a button;
- *Button_GetTextMargin* - Gets the margins used to draw text in a button control. You can use this macro or send the BCM_GETTEXTMARGIN message explicitly;
- *Button_SetCheck* - Sets the check state of a radio button or check box. You can use this macro or send the BM_SETCHECK message explicitly;
- *Button_SetDropDownState* - Sets the drop down state for a specified button with style of BS_SPLITBUTTON. Use this macro or send the BCM_SETDROPDOWNSTATE message explicitly;
- *Button_SetElevationRequiredState* - Sets the elevation required state for a specified button or command link to display an elevated icon. Use this macro or send the BCM_SETSHIELD message explicitly.
- *Button_SetImageList* - Assigns an image list to a button control. You can use this macro or send the BCM_SETIMAGELIST message explicitly.
- *Button_SetNote* - Sets the text of the note associated with a specified command link button. You can use this macro or send the BCM_SETNOTE message explicitly;

- *Button_SetSplitInfo* - Sets information for a specified split button control. Use this macro or send the BCM_SETSPLITINFO message explicitly;
- *Button_SetState* - Sets the highlight state of a button. The highlight state indicates whether the button is highlighted as if the user had pushed it. You can use this macro or send the BM_SETSTATE message explicitly;
- *Button_SetStyle* - Sets the style of a button. You can use this macro or send the BM_SETSTYLE message explicitly;
- *Button_SetText* - Sets the text of a button;
- *Button_SetTextMargin* - Sets the margins for drawing text in a button control. You can use this macro or send the BCM_SETTEXTMARGIN message explicitly;

- ***Combo Box***

A *combo box* is a unique type of control, defined by the COMBOBOX class, that combines much of the functionality of a list box and an edit control.

Functions:

- *DlgDirListComboBox* - Replaces the contents of a combo box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list of names can include mapped drive letters;
- *DlgDirSelectComboBoxEx* - Retrieves the current selection from a combo box filled by using the DlgDirListComboBox function. The selection is interpreted as a drive letter, a file, or a directory name;

- *GetComboBoxInfo* - Retrieves information about the specified combo box;

Macros:

- *ComboBox_AddItemData* - Adds item data to the list in a combo box at the specified location. You can use this macro or send the CB_ADDSTRING message explicitly;
- *ComboBox_AddString* - Adds a string to a list in a combo box. If the combo box does not have the CBS_SORT style, the string is added to the end of the list. Otherwise, the string is inserted into the list and the list is sorted. You can use this macro or send the CB_ADDSTRING message explicitly;
- *ComboBox_DeleteString* - Deletes the item at the specified location in a list in a combo box. You can use this macro or send the CB_DELETESTRING message explicitly;
- *ComboBox_Dir* - Adds names to the list displayed by a combo box. The macro adds the names of directories and files that match a specified string and set of file attributes. It can also add mapped drive letters to the list in a combo box. You can use this macro or send the CB_DIR message explicitly;
- *ComboBox_Enable* - Enables or disables a combo box control;
- *ComboBox_FindItemData* - Finds the first item in a combo box list that has the specified item data. You can use this macro or send the CB_FINDSTRING message explicitly;
- *ComboBox_FindString* - Finds the first string in a combo box list that begins with the specified string. You can use this macro or send the CB_FINDSTRING message explicitly;

- *ComboBox_FindStringExact* - Finds the first string in a combo box list that exactly matches the specified string, except that the search is not case sensitive. You can use this macro or send the CB_FINDSTRINGEXACT message explicitly;
- *ComboBox_GetCount* - Gets the number of items in the list box of a combo box. You can use this macro or send the CB_GETCOUNT message explicitly;
- *ComboBox_GetCueBannerText* - Gets the cue banner text displayed in the edit control of a combo box. Use this macro or send the CB_GETCUEBANNER message explicitly;
- *ComboBox_GetCurSel* - Gets the index of the currently selected item in a combo box. You can use this macro or send the CB_GETCURSEL message explicitly;
- *ComboBox_GetDroppedControlRect* - Retrieves the screen coordinates of a combo box in its dropped-down state. You can use this macro or send the CB_GETDROPPEDCONTROLRECT message explicitly;
- *ComboBox_GetDroppedState* - Ascertains whether the drop list in a combo box control is visible. You can use this macro or send the CB_GETDROPPEDSTATE message explicitly;
- *ComboBox_GetExtendedUI* - Ascertains whether a combo box is using the default user interface (UI) or the extended UI. You can use this macro or send the CB_GETEXTENDEDUI message explicitly;
- *ComboBox_GetItemData* - Gets the application-defined value associated with the specified list item in a combo box. You can use this macro or send the CB_GETITEMDATA message explicitly;

- *ComboBox_GetItemHeight* - Retrieves the height of list items in a combo box. You can use this macro or send the CB_GETITEMHEIGHT message explicitly;
- *ComboBox_GetLBText* - Gets a string from a list in a combo box. You can use this macro or send the CB_GETLBTEXT message explicitly;
- *ComboBox_GetLBTextLen* - Gets the length of a string in the list in a combo box. You can use this macro or send the CB_GETLBTEXTLEN message explicitly;
- *ComboBox_GetMinVisible* - Gets the minimum number of visible items in the drop-down list of a combo box;
- *ComboBox_GetText* - Retrieves the text from a combo box control;
- *ComboBox_GetTextLength* - Gets the number of characters in the text of a combo box;
- *ComboBox_InsertItemData* - Inserts item data in a list in a combo box at the specified location. You can use this macro or send the CB_INSERTSTRING message explicitly;
- *ComboBox_InsertString* - Adds a string to a list in a combo box at the specified location. You can use this macro or send the CB_INSERTSTRING message explicitly;
- *ComboBox_LimitText* - Limits the length of the text the user may type into the edit control of a combo box. You can use this macro or send the CB_LIMITTEXT message explicitly;
- *ComboBox_ResetContent* - Removes all items from the list box and edit control of a combo box. You can use this macro or send the CB_RESETCONTENT message explicitly;

- *ComboBox_SelectItemData* - Searches a list in a combo box for an item that has the specified item data. If a matching item is found, the item is selected. You can use this macro or send the CB_SELECTSTRING message explicitly;
- *ComboBox_SelectString* - Searches a list in a combo box for an item that begins with the characters in a specified string. If a matching item is found, the item is selected. You can use this macro or send the CB_SELECTSTRING message explicitly;
- *ComboBox_SetCueBannerText* - Sets the cue banner text that is displayed for the edit control of a combo box;
- *ComboBox_SetCurSel* - Sets the currently selected item in a combo box. You can use this macro or send the CB_SETCURSEL message explicitly;
- *ComboBox_SetExtendedUI* - Selects either the default user interface (UI) or the extended UI for a combo box that has the CBS_DROPDOWN or CBS_DROPDOWNLIST style. You can use this macro or send the CB_SETEXTENDEDUI message explicitly;
- *ComboBox_SetItemData* - Sets the application-defined value associated with the specified list item in a combo box. You can use this macro or send the CB_SETITEMDATA message explicitly;
- *ComboBox_SetItemHeight* - Sets the height of list items or the selection field in a combo box. You can use this macro or send the CB_SETITEMHEIGHT message explicitly;
- *ComboBox_SetMinVisible* - Sets the minimum number of visible items in the drop-down list of a combo box;
- *SetText* - Sets the text of a combo box;

- *ComboBox_ShowDropdown* - Shows or hides the list in a combo box. You can use this macro or send the CB_RESETCONTENT message explicitly;
- *ListBox_AddItemData* - Adds item data to the list box at the specified location. You can use this macro or send the LB_ADDSTRING message explicitly;

- ***Combo BoxEx***

ComboBoxEx controls are combo box controls that provide native support for item images. To make item images easily accessible, the control provides image list support. By using this control, you can provide the functionality of a combo box without having to manually draw item graphics.

- ***Edit Control***

An *edit control* is a rectangular control window typically used in a dialog box to permit the user to enter and edit text by typing on the keyboard.

Functions:

- *EditWordBreakProc* - An application-defined callback function used with the EM_SETWORDBREAKPROC message. A multiline edit control or a rich edit control calls an EditWordBreakProc function to break a line of text;
- *EditWordBreakProc* - is a placeholder for the application-defined function name;

Macros:

- *Edit_CanUndo* **-** Determines whether there are any actions in the undo queue of an edit or rich edit control. You can use this macro or send the EM_CANUNDO message explicitly;
- *Edit_EmptyUndoBuffer* **-** Resets the undo flag of an edit or rich edit control. The undo flag is set whenever an operation within the edit control can be undone. You can use this macro or send the EM_EMPTYUNDOBUFFER message explicitly;
- *Edit_Enable* **-** Enables or disables an edit control;
- *Edit_FormatLines* **-** Sets a flag that determines whether text retrieved from a multiline edit control includes soft line-break characters. A soft line break consists of two carriage returns and a line feed and is inserted at the end of a line that is broken because of wordwrapping. You can use this macro or send the EM_FMTLINES message explicitly;
- *Edit_GetCueBannerText* **-** Gets the text that is displayed as a textual cue, or tip, in an edit control. You can use this macro or send the EM_GETCUEBANNER message explicitly;
- *Edit_GetFirstVisibleLine* **-** Gets the index of the uppermost visible line in a multiline edit or rich edit control. You can use this macro or send the EM_GETFIRSTVISIBLELINE message explicitly;
- *Edit_GetHandle* **-** Gets a handle to the memory currently allocated for the text of a multiline edit control. You can use this macro or send the EM_GETHANDLE message explicitly;
- *Edit_GetHilite* **-** Not implemented;

- *Edit_GetLine* - Retrieves a line of text from an edit or rich edit control. You can use this macro or send the EM_GETLINE message explicitly;
- *Edit_GetLineCount* - Gets the number of lines in the text of an edit control. You can use this macro or send the EM_GETLINECOUNT message explicitly;
- *Edit_GetModify* - Gets the state of an edit or rich edit control's modification flag. The flag indicates whether the contents of the control have been modified. You can use this macro or send the EM_GETMODIFY message explicitly;
- *Edit_GetPasswordChar* - Gets the password character for an edit or rich edit control. You can use this macro or send the EM_GETPASSWORDCHAR message explicitly;
- *Edit_GetRect* - Gets the formatting rectangle of an edit control. You can use this macro or send the EM_GETRECT message explicitly.
- *Edit_GetSel* - Gets the starting and ending character positions of the current selection in an edit or rich edit control. You can use this macro or send the EM_GETSEL message explicitly;
- *Edit_GetText* - Gets the text of an edit control;
- *Edit_GetTextLength* - Gets the number of characters in the text of an edit control;
- *Edit_GetWordBreakProc* - Retrieves the address of an edit or rich edit control's Wordwrap function. You can use this macro or send the EM_GETWORDBREAKPROC message explicitly;
- *Edit_HideBalloonTip* - Hides any balloon tip associated with an edit control. You can use this macro or send the EM_HIDEBALLOONTIP message explicitly;

- *Edit_LimitText* - Limits the length of text that can be entered into an edit control. You can use this macro or send the EM_LIMITTEXT message explicitly;
- *Edit_LineFromChar* - Gets the index of the line that contains the specified character index in a multiline edit or rich edit control. You can use this macro or send the EM_LINEFROMCHAR message explicitly;
- *Edit_LineIndex* - Gets the character index of the first character of a specified line in a multiline edit or rich edit control. You can use this macro or send the EM_LINEINDEX message explicitly;
- *Edit_LineLength* - Retrieves the length, in characters, of a line in an edit or rich edit control. You can use this macro or send the EM_LINELENGTH message explicitly;
- *Edit_ReplaceSel* - Replaces the selected text in an edit control or a rich edit control with the specified text. You can use this macro or send the EM_REPLACESEL message explicitly;
- *Edit_Scroll* - Scrolls the text vertically in a multiline edit or rich edit control. You can use this macro or send the EM_SCROLL message explicitly;
- *Edit_ScrollCaret* - Scrolls the caret into view in an edit or rich edit control. You can use this macro or send the EM_SCROLLCARET message explicitly;
- *Edit_SetCueBannerText* - Sets the text that is displayed as the textual cue, or tip, for an edit control. You can use this macro or send the EM_SETCUEBANNER message explicitly;

- *Edit_SetCueBannerTextFocused* - Sets the text that is displayed as the textual cue, or tip, for an edit control. You can use this macro or send the EM_SETCUEBANNER message explicitly;
- *Edit_SetHandle* - Sets the handle of the memory that will be used by a multiline edit control. You can use this macro or send the EM_SETHANDLE message explicitly;
- *Edit_SetHilite* - Not implemented;
- *Edit_SetModify* - Sets or clears the modification flag for an edit control. The modification flag indicates whether the text within the edit control has been modified. You can use this macro or send the EM_SETMODIFY message explicitly;
- *Edit_SetPasswordChar* - Sets or removes the password character for an edit or rich edit control. When a password character is set, that character is displayed in place of the characters typed by the user. You can use this macro or send the EM_SETPASSWORDCHAR message explicitly;
- *Edit_SetReadOnly* - Sets or removes the read-only style (ES_READONLY) of an edit or rich edit control. You can use this macro or send the EM_SETREADONLY message explicitly;
- *Edit_SetRect* - Sets the formatting rectangle of an edit control. You can use this macro or send the EM_SETRECT message explicitly;
- *Edit_SetRectNoPaint* - Sets the formatting rectangle of a multiline edit control. This macro is equivalent to Edit_SetRect, except that it does not redraw the edit control window. You can use this macro or send the EM_SETRECTNP message explicitly;

- *Edit_SetSel* - Selects a range of characters in an edit or rich edit control. You can use this macro or send the EM_SETSEL message explicitly;
- *Edit_SetTabStops* - Sets the tab stops in a multiline edit or rich edit control. When text is copied to the control, any tab character in the text causes space to be generated up to the next tab stop. You can use this macro or send the EM_SETTABSTOPS message explicitly.
- *Edit_SetText* - Sets the text of an edit control;
- *Edit_SetWordBreakProc* - Replaces an edit control's default Wordwrap function with an application-defined Wordwrap function. You can use this macro or send the EM_SETWORDBREAKPROC message explicitly;
- *Edit_ShowBalloonTip* - Displays a balloon tip associated with an edit control. You can use this macro or send the EM_SHOWBALLOONTIP message explicitly;
- *Edit_Undo* - Undoes the last operation in the undo queue of an edit or rich edit control. You can use this macro or send the EM_UNDO message explicitly;

- ***List Box***

A list box is a control window that contains a simple list of items from which the user can choose.

Functions:

- *DlgDirList* - Replaces the contents of a list box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list can optionally include mapped drives;

- *DlgDirSelectEx* - Retrieves the current selection from a single-selection list box. It assumes that the list box has been filled by the DlgDirList function and that the selection is a drive letter, filename, or directory name;
- *GetListBoxInfo* - Retrieves information about the specified list box;

Messages:

- *LB_ADDFILE* - Adds the specified filename to a list box that contains a directory listing;
- *LB_ADDSTRING* - Adds a string to a list box. If the list box does not have the LBS_SORT style, the string is added to the end of the list. Otherwise, the string is inserted into the list and the list is sorted;
- *LB_DELETESTRING* - Deletes a string in a list box;
- *LB_DIR* - Adds names to the list displayed by a list box. The message adds the names of directories and files that match a specified string and set of file attributes. LB_DIR can also add mapped drive letters to the list box;
- *LB_FINDSTRING* - Finds the first string in a list box that begins with the specified string;
- *LB_FINDSTRINGEXACT* - Finds the first list box string that exactly matches the specified string, except that the search is not case sensitive;
- *LB_GETANCHORINDEX* - Gets the index of the anchor item
- *LB_GETCARETINDEX* - Retrieves the index of the item that has the focus rectangle in a multiple-selection list box. The item may or may not be selected;

- *LB_GETCOUNT* - Gets the number of items in a list box;
- *LB_GETCURSEL* - Gets the index of the currently selected item, if any, in a single-selection list box;
- *LB_GETHORIZONTALEXTENT* - Gets the width, in pixels, that a list box can be scrolled horizontally (the scrollable width) if the list box has a horizontal scroll bar;
- *LB_GETITEMDATA* - Gets the application-defined value associated with the specified list box item;
- *LB_GETITEMHEIGHT* - Gets the height of items in a list box;
- *LB_GETITEMRECT* - Gets the dimensions of the rectangle that bounds a list box item as it is currently displayed in the list box;
- *LB_GETLISTBOXINFO* - Gets the number of items per column in a specified list box;
- *LB_GETLOCALE* - Gets the current locale of the list box. You can use the locale to determine the correct sorting order of displayed text (for list boxes with the LBS_SORT style) and of text added by the LB_ADDSTRING message;
- *LB_GETSEL* - Gets the selection state of an item;
- *LB_GETSELCOUNT* - Gets the total number of selected items in a multiple-selection list box;
- *LB_GETSELITEMS* - Fills a buffer with an array of integers that specify the item numbers of selected items in a multiple-selection list box;
- *LB_GETTEXT* - Gets a string from a list box;
- *LB_GETTEXTLEN* - Gets the length of a string in a list box;
- *LB_GETTOPINDEX* - Gets the index of the first visible item in a list box. Initially the item with index 0 is at the top of the list box, but if

the list box contents have been scrolled another item may be at the top;

- *LB_INITSTORAGE* - Allocates memory for storing list box items. This message is used before an application adds a large number of items to a list box;
- *LB_INSERTSTRING* - Inserts a string or item data into a list box. Unlike the LB_ADDSTRING message, the LB_INSERTSTRING message does not cause a list with the LBS_SORT style to be sorted;
- *LB_ITEMFROMPOINT* - Gets the zero-based index of the item nearest the specified point in a list box;
- *LB_RESETCONTENT* - Removes all items from a list box;
- *LB_SELECTSTRING* - Searches a list box for an item that begins with the characters in a specified string. If a matching item is found, the item is selected;
- *LB_SELITEMRANGE* - Selects or deselects one or more consecutive items in a multiple-selection list box;
- *LB_SELITEMRANGEEX* - Selects one or more consecutive items in a multiple-selection list box;
- *LB_SETANCHORINDEX* - Sets the anchor item;
- *LB_SETCARETINDEX* - Sets the focus rectangle to the item at the specified index in a multiple-selection list box. If the item is not visible, it is scrolled into view;
- *LB_SETCOLUMNWIDTH* - Sets the width, in pixels, of all columns in a multiple-column list box;
- *LB_SETCOUNT* - Sets the count of items in a list box created with the LBS_NODATA style and not created with the LBS_HASSTRINGS style;

- *LB_SETCURSEL* - Selects a string and scrolls it into view, if necessary. When the new string is selected, the list box removes the highlight from the previously selected string;
- *LB_SETHORIZONTALEXTENT* - Sets the width, in pixels, by which a list box can be scrolled horizontally (the scrollable width). If the width of the list box is smaller than this value, the horizontal scroll bar horizontally scrolls items in the list box. If the width of the list box is equal to or greater than this value, the horizontal scroll bar is hidden;
- *LB_SETITEMDATA* - Sets a value associated with the specified item in a list box;
- *LB_SETITEMHEIGHT* - Sets the height, in pixels, of items in a list box. If the list box has the LBS_OWNERDRAWVARIABLE style, this message sets the height of the item specified by the *wParam* parameter. Otherwise, this message sets the height of all items in the list box;
- *LB_SETLOCALE* - Sets the current locale of the list box. You can use the locale to determine the correct sorting order of displayed text (for list boxes with the LBS_SORT style) and of text added by the LB_ADDSTRING message;
- *LB_SETSEL* - Selects a string in a multiple-selection list box;
- *LB_SETTABSTOPS* - Sets the tab-stop positions in a list box;
- *LB_SETTOPINDEX* - Ensures that the specified item in a list box is visible;

- http://www.functionx.com/vcsharp/index.htm
- http://msdn.microsoft.com/en-us/library/bb773173%28VS.85%29.aspx

3. Class work

Ex1: Create a window with a title and some text in it.

Ex2: Create a window with a button in it, and a counter which counts the number of times the user has clicked the button.

Ex3: Create a simple calculator (Figure 1).

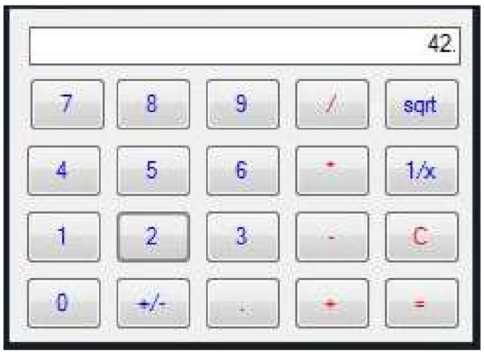


Figure 19: Simple Calculator

Ex4: Create a **X-0** game (Figure 2).

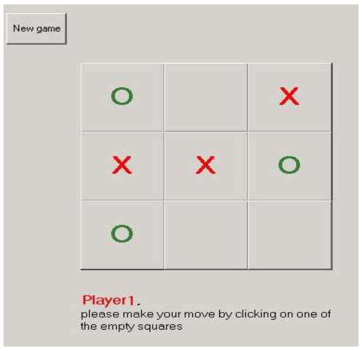

New game
O
X
X
X
O
O
Player1 ,
please make your move by clicking on one of the empty squares

Figure 20: X-0 game

4. Home Work

Ex1: Create a game called Snake through C#. Concept is to create a snake which move around a given screen according to user input. It eats objects randomly emerging on screen and if succesfull in eating them it becomes larger in size and gains score. When the snake dies, the game is over. The snake dies when hits an obstacle or hit its own body.

Lesson 6

Java Web Services

1. Introduction

Web services today are frequently just *Application Programming Interfaces (API)* or *Web APIs* that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. In common usage the term refers to clients and servers that communicate over the Hypertext Transfer Protocol (HTTP) protocol used on the web. Such services tend to fall into one of two camps: Big Web Services and RESTful Web Services.

Big Web Services" use Extensible Markup Language (XML) messages that follow the Simple Object Access Protocol (SOAP) standard and have been popular with traditional enterprise. In such systems, there is often a machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP *endpoint*, but it is a prerequisite for automated client-side code generation in many Java and .NET SOAP frameworks (frameworks such as Spring, Apache Axis2 and Apache CXF being notable exceptions). Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a web service.

More recently, *REpresentational State Transfer (RESTful)* web services have been regaining popularity, particularly with Internet companies. By using the PUT, GET and DELETE HTTP methods, alongside POST, these are often better integrated with HTTP and web browsers than SOAP-based services. They do not require XML messages or WSDL service-API definitions.

2. Web service applications with NetBeans IDE

The Java API for XML Web Services (JAX-WS) is the current model for SOAP-based web services in Metro. JAX-WS is built on the earlier JAX-RPC model but uses specific Java EE 5 features, such as annotations, to simplify the task of developing web services. Because it uses SOAP for messaging, JAX-WS is transport neutral. It also supports a wide range of modular WS-* specifications, such as WS-Security and WS-ReliableMessaging.

When you create a web service client, you have the option of using either the JAX-WS or JAX-RPC model. This is because some older JAX-RPC services use a binding style that is not supported by JAX-WS. These services can only be consumed by JAX-RPC clients.

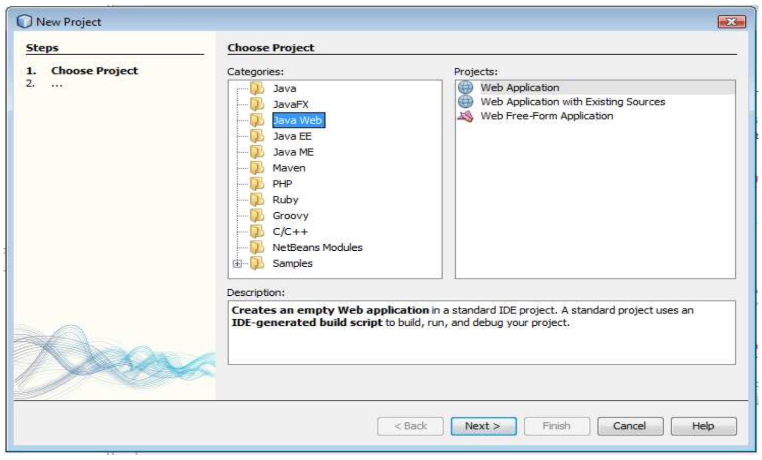
Metro web services are interoperable with Apache Axis2 web services. Apache Axis2 is an open-source implementation of the SOAP (Simple Object-Access Protocol) submission to the W3C. Axis2 not only supports SOAP 1.1 and SOAP 1.2, but it also has integrated support for RESTful web services.

2.1. Creating a Web Service

Step 1: Choosing a Container

You can either deploy your web service in a web container or in an EJB container. This depends on your choice of implementation. If you are creating a Java EE 6 application, use a web container in any case, because you can put EJBs directly in a web application. For example, if you plan to deploy to the Tomcat Web Server, which only has a web container, create a web application, not an EJB module.

- Choose File > New Project (Ctrl-Shift-N). Select Web Application from the Java Web category or EJB Module from the Java EE category;



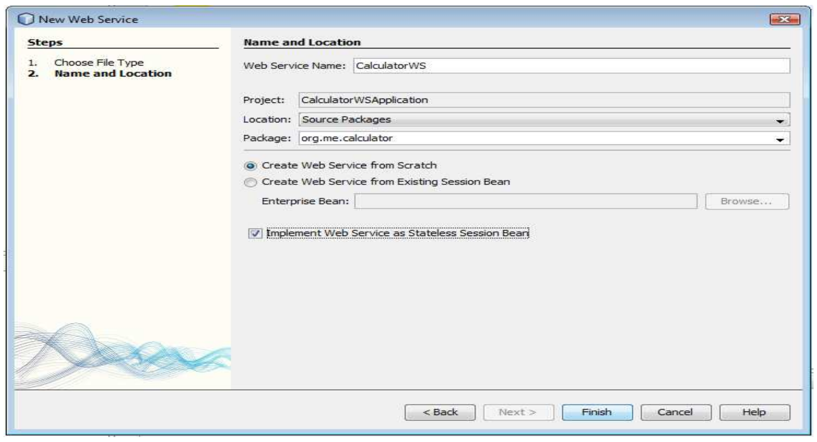
Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

- Name the project CalculatorWSApplication;
- Click through the remaining pages and click Finish;

Step 2: Creating a Web Service from a Java Class

- Right-click the CalculatorWSApplication node and choose New -> Web Service;
- Name the web service CalculatorWS and type org.me.calculator in Package;

- If you are creating a Java EE 6 project, leave Create Web Service from Scratch selected, and select Implement Web Service as a Stateless Session Bean;



Name and Location

Web Service Name: CalculatorWS

Project: CalculatorWSApplication

Location: Source Packages

Package: org.me.calculator

(●) Create Web Service from Scratch

( ) Create Web Service from Existing Session Bean

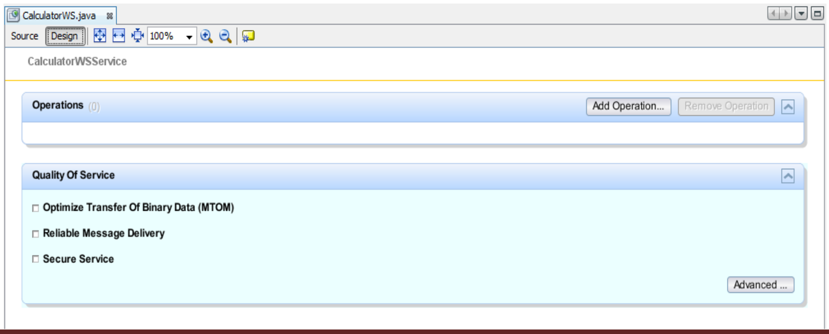
Enterprise Bean:

[✓] Implement Web Service as Stateless Session Bean

- Click Finish. The Projects window displays the structure of the new web service and the source code is shown in the editor area;

2.2. Designing the Web Service - Adding an Operation to the Web Service

- Change to the Design view;

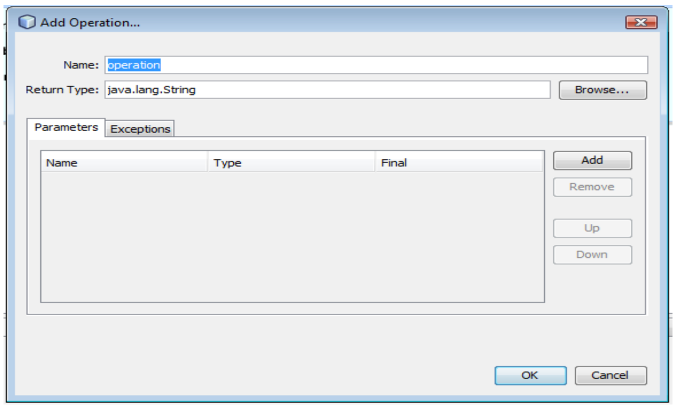


[ ] Optimize Transfer Of Binary Data (MTOM)

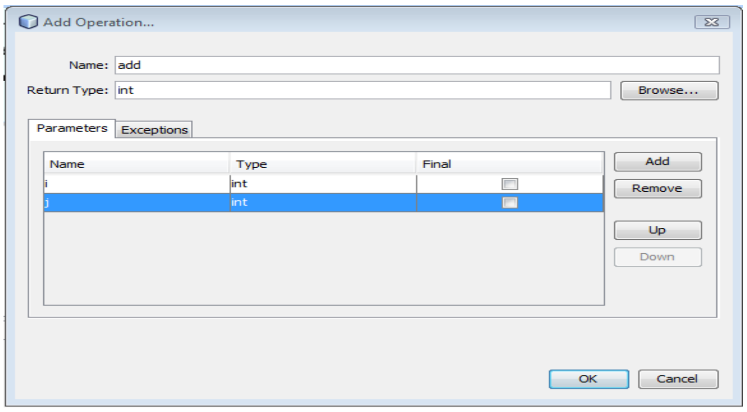
[ ] Reliable Message Delivery

[ ] Secure Service

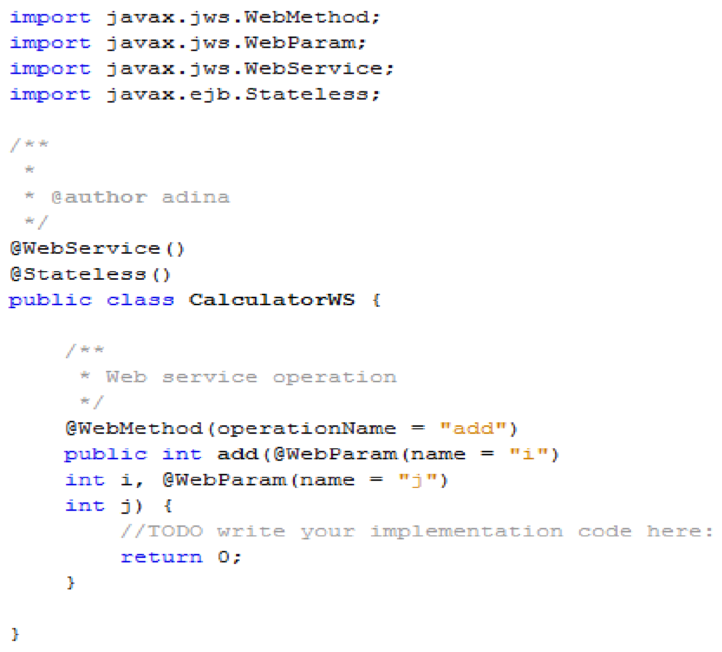
- Click Add Operation in the visual designer. A dialog box appears where you can define the new operation;



- In the upper part of the Add Operation dialog box, type add in Name and type int in the Return Type drop-down list. In the lower part of the Add Operation dialog box, click Add and create a parameter of type int named i. Then click Add again and create a parameter of type int called j;

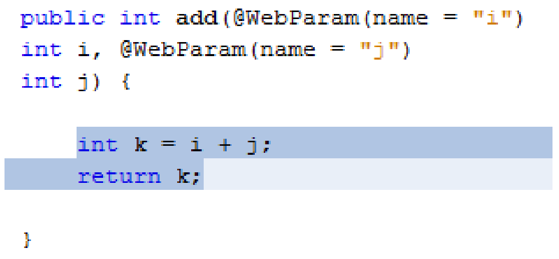


- Click OK at the bottom of the Add Operation dialog box;
- Click Source and view the code that you generated in the previous steps;



```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.ejb.Stateless;

/**
 *
 * @author adina
 */
@WebService()
@Stateless()
public class CalculatorWS {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i")
    int i, @WebParam(name = "j")
    int j) {
        //TODO write your implementation code here:
        return 0;
    }

}
```

- In the editor, extend the skeleton add operation to the following;

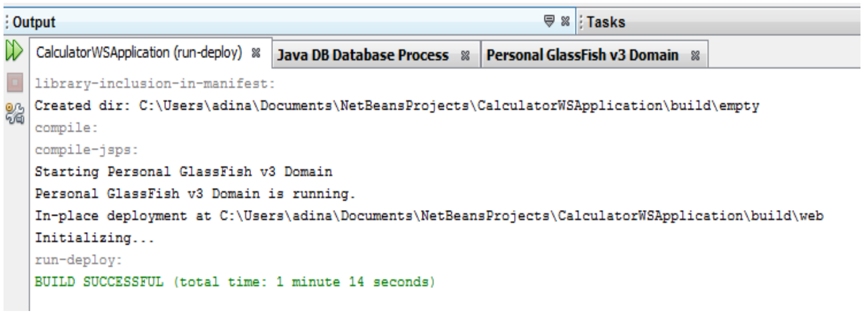


```
public int add(@WebParam(name = "i")
int i, @WebParam(name = "j")
int j) {

    int k = i + j;
    return k;

}
```

2.3. Deploying and Testing the Web Service

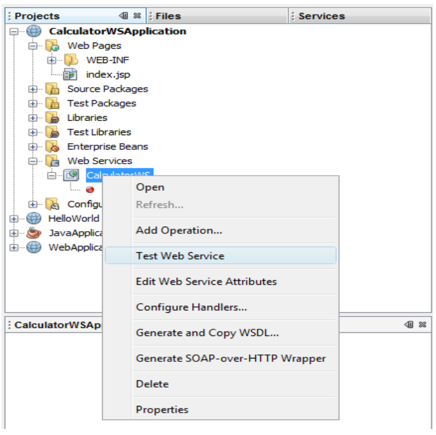
When you deploy a web service to a web container, the IDE lets you test the web service to see if it functions as you expect. The Tester application, provided by GlassFish, is integrated into the IDE for this purpose. For the Tomcat Web Server, there is a similar tool. However, while GlassFish's Tester page lets you enter values and test them, the Tomcat Web Server does not. In the latter case, you can only see that the web service is deployed, you cannot test the values.

- Right-click the project and choose Deploy. The IDE starts the application server, builds the application, and deploys the application to the server. You can follow the progress of these operations in the CalculatorWSApplication (run-deploy) and GlassFish or Tomcat tabs in the Output view;

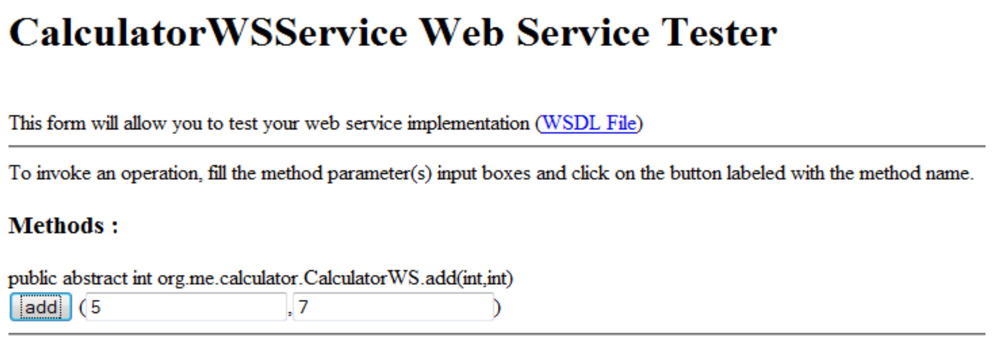

Output
Tasks
CalculatorWSApplication (run-deploy)
Java DB Database Process
Personal GlassFish v3 Domain

```
library-inclusion-in-manifest:
Created dir: C:\Users\adina\Documents\NetBeansProjects\CalculatorWSApplication\build\empty
compile:
compile-jsps:
Starting Personal GlassFish v3 Domain
Personal GlassFish v3 Domain is running.
In-place deployment at C:\Users\adina\Documents\NetBeansProjects\CalculatorWSApplication\build\web
Initializing...
run-deploy:
BUILD SUCCESSFUL (total time: 1 minute 14 seconds)
```

- In the IDE's Projects tab, expand the Web Services node of the CalculatorWSApplication project. Right-click the CalculatorWS node, and choose Test Web Service;



The IDE opens the tester page in your browser, if you deployed a web application to GlassFish:



CalculatorWSService Web Service Tester

This form will allow you to test your web service implementation (WSDL File)

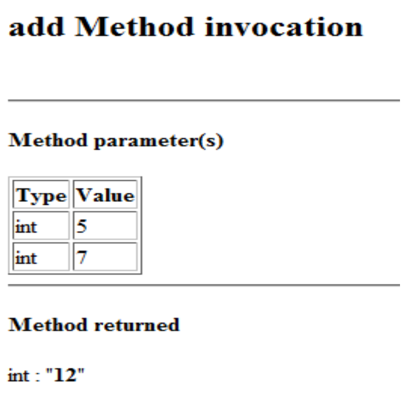
To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract int org.me.calculator.CalculatorWS.add(int,int)

add (5 , 7)

The sum:



add Method invocation

Method parameter(s)

Type	Value
int	5
int	7

Method returned

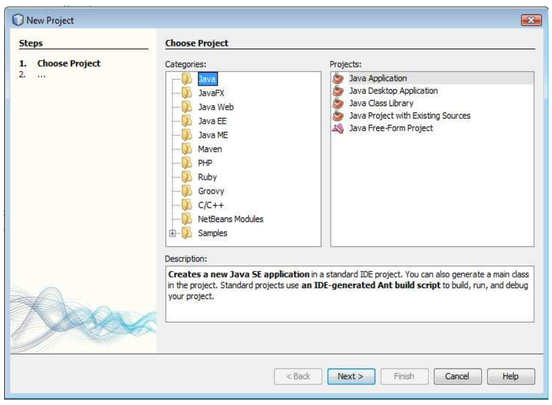
int : "12"

2.4.Consuming the Web Service

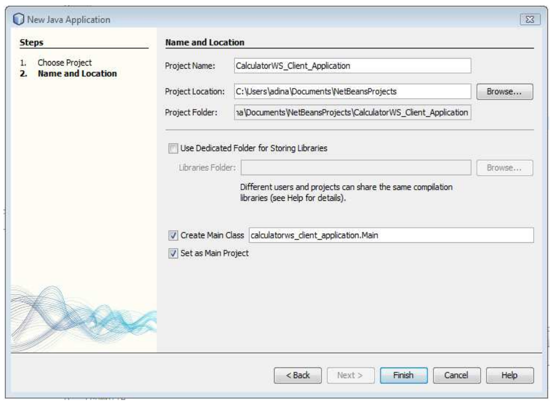
Now that you have deployed the web service, you need to create a client to make use of the web service's add method. Here, you create three clients— a Java class in a Java SE application, a servlet, and a JSP page in a web application.

Client 1: Java Class in Java SE Application

- Choose File -> New Project (Ctrl-Shift-N). Select Java Application from the Java category. Name the project CalculatorWS_Client_Application. Leave Create Main Class selected and accept all other default settings. Click Finish;

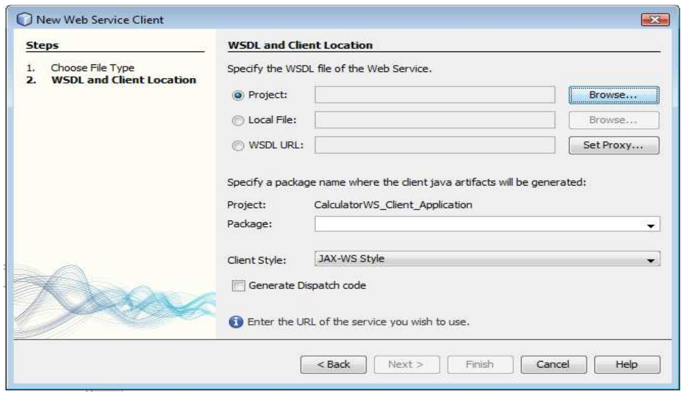


Description:
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.



Project Name: CalculatorWS_Client_Application
Project Location: C:\Users\adina\Documents\NetBeansProjects
Project Folder: ıa\Documents\NetBeansProjects\CalculatorWS_Client_Application
☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).
☑ Create Main Class calculatorws_client_application.Main
☑ Set as Main Project

- Right-click the CalculatorWS_Client_Application node and choose New -> Web Service Client;



Specify the WSDL file of the Web Service.

Project:

Local File:

WSDL URL:

Specify a package name where the client java artifacts will be generated:

Project: CalculatorWS_Client_Application

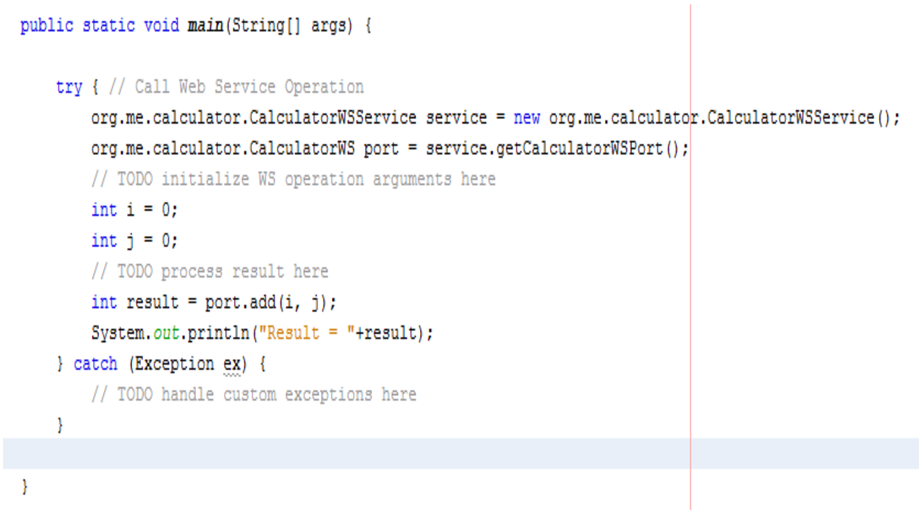
Package:

Client Style: JAX-WS Style

Generate Dispatch code

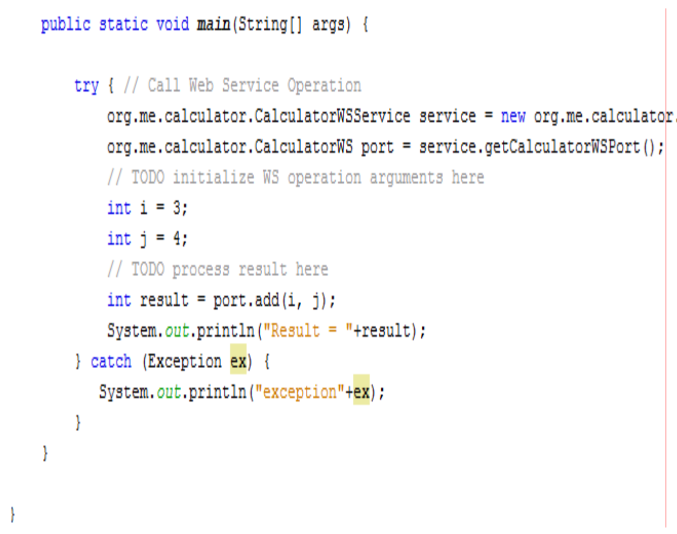
Enter the URL of the service you wish to use.

- In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK;
- Leave the other settings at default and click Finish;
- Right-click in the editor and then choose Insert Code > Call Web Service Operation;



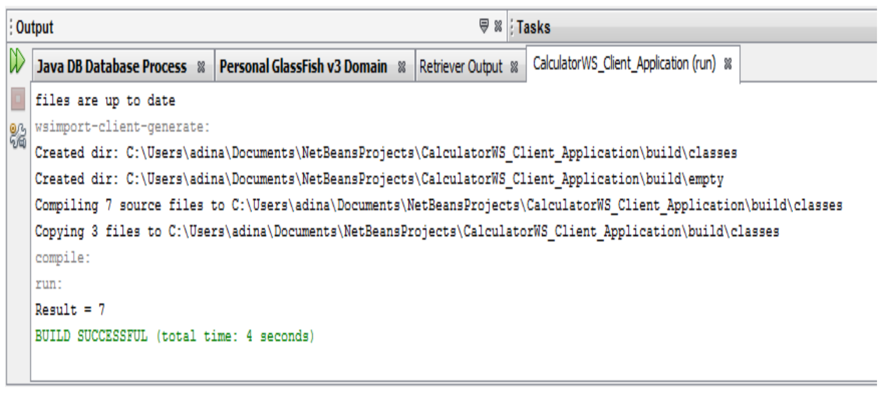
```
public static void main(String[] args) {

    try { // Call Web Service Operation
        org.me.calculator.CalculatorWSService service = new org.me.calculator.CalculatorWSService();
        org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
        // TODO initialize WS operation arguments here
        int i = 0;
        int j = 0;
        // TODO process result here
        int result = port.add(i, j);
        System.out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }

}
```

- Initialize the two ints. Just change the values of the two ints above from 0 to other integers, such as 3 and 4;
- In the catch block, replace the commented out TODO with System.out.println("exception" + ex);



```
    public static void main(String[] args) {

        try { // Call Web Service Operation
            org.me.calculator.CalculatorWSService service = new org.me.calculator.CalculatorWSService();
            org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
            // TODO initialize WS operation arguments here
            int i = 3;
            int j = 4;
            // TODO process result here
            int result = port.add(i, j);
            System.out.println("Result = "+result);
        } catch (Exception ex) {
           System.out.println("exception"+ex);
        }
    }

}
```

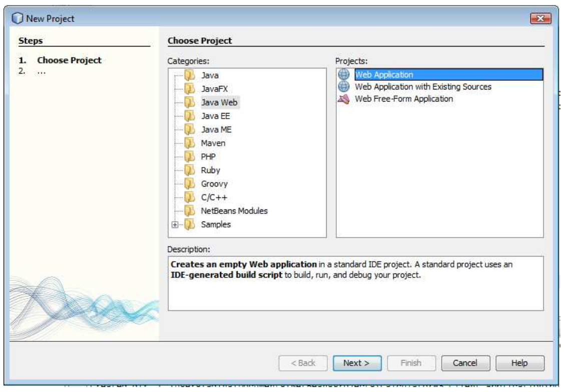
- Right-click the project node and choose Run;


Output
Tasks
Java DB Database Process
Personal GlassFish v3 Domain
Retriever Output
CalculatorWS_Client_Application (run)

```
files are up to date
wsimport-client-generate:
Created dir: C:\Users\adina\Documents\NetBeansProjects\CalculatorWS_Client_Application\build\classes
Created dir: C:\Users\adina\Documents\NetBeansProjects\CalculatorWS_Client_Application\build\empty
Compiling 7 source files to C:\Users\adina\Documents\NetBeansProjects\CalculatorWS_Client_Application\build\classes
Copying 3 files to C:\Users\adina\Documents\NetBeansProjects\CalculatorWS_Client_Application\build\classes
compile:
run:
Result = 7
BUILD SUCCESSFUL (total time: 4 seconds)
```

Client 2: Servlet in Web Application

- Choose File -> New Project (Ctrl-Shift-N). Select Web Application from the Java Web category. Name the project CalculatorWSServletClient. Click Next and then click Finish;


New Project

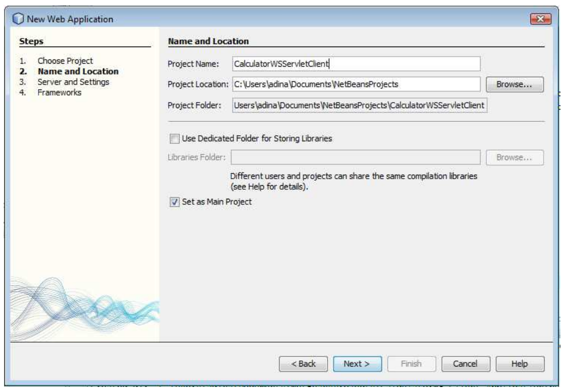
Steps
1. Choose Project
2. ...

Choose Project

Categories: Java, JavaFX, Java Web, Java EE, Java ME, Maven, PHP, Ruby, Groovy, C/C++, NetBeans Modules, Samples

Projects: Web Application (selected), Web Application with Existing Sources, Web Free-Form Application

Description:
Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.


New Web Application

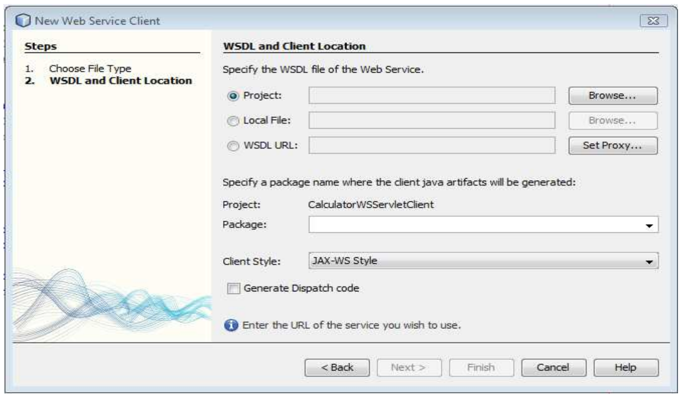
Steps
1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name: CalculatorWSServletClient
Project Location: C:\Users\adina\Documents\NetBeansProjects
Project Folder: Users\adina\Documents\NetBeansProjects\CalculatorWSServletClient

[ ] Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).
[✓] Set as Main Project

- Right-click the CalculatorWSServletClient node and choose New -> Web Service Client. The New Web Service Client wizard appears;


New Web Service Client

Steps
1. Choose File Type
2. **WSDL and Client Location**

WSDL and Client Location

Specify the WSDL file of the Web Service.
(●) Project:
( ) Local File:
( ) WSDL URL:

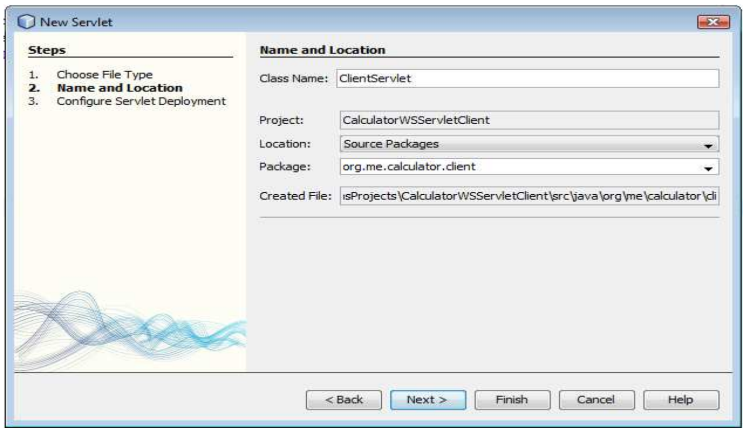
Specify a package name where the client java artifacts will be generated:
Project: CalculatorWSServletClient
Package:

Client Style: JAX-WS Style
[ ] Generate Dispatch code

Enter the URL of the service you wish to use.

- In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK;

- Leave the other settings at default and click Finish;
- Right-click the CalculatorWSServletClient project node and choose New -> Servlet. Name the servlet ClientServlet and place it in a package called org.me.calculator.client. Click Finish;



New Servlet

Steps
1. Choose File Type
2. **Name and Location**
3. Configure Servlet Deployment

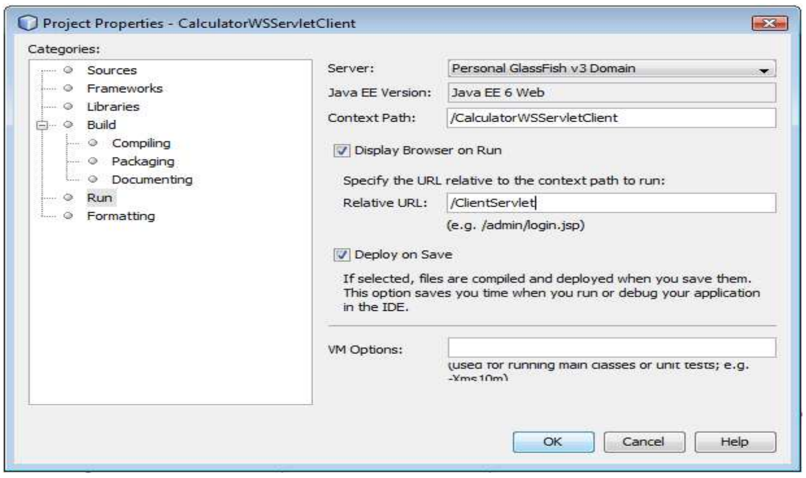
Name and Location

Class Name: ClientServlet

Project: CalculatorWSServletClient
Location: Source Packages
Package: org.me.calculator.client

Created File: sProjects\CalculatorWSServletClient\src\java\org\me\calculator\cli

- To make the servlet the entry point to your application, right-click the CalculatorWSServletClient project node and choose Properties. Open the Run properties and type /ClientServlet in the Relative URL field. Click OK.



Project Properties - CalculatorWSServletClient

Categories:
- Sources
- Frameworks
- Libraries
- Build
  - Compiling
  - Packaging
  - Documenting
- Run
- Formatting

Server: Personal GlassFish v3 Domain
Java EE Version: Java EE 6 Web
Context Path: /CalculatorWSServletClient

[✓] Display Browser on Run

Specify the URL relative to the context path to run:
Relative URL: /ClientServlet
(e.g. /admin/login.jsp)

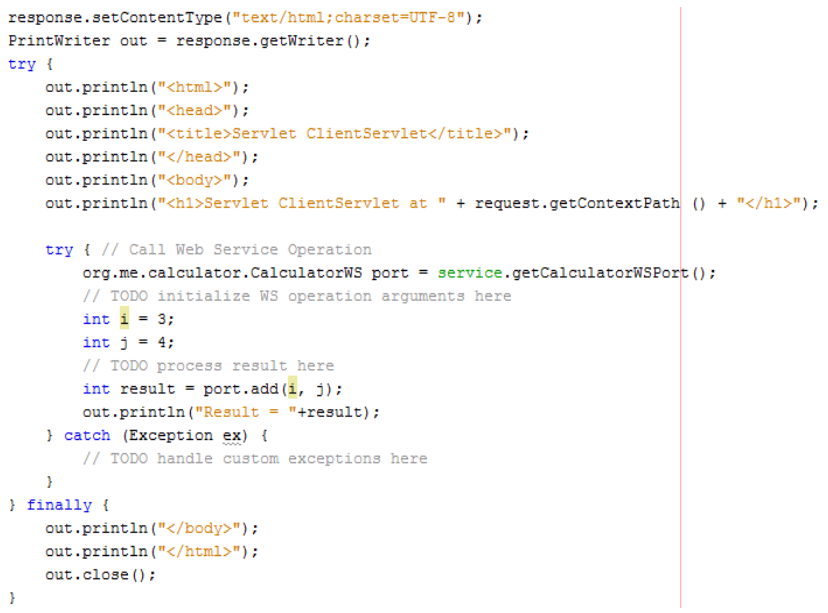
[✓] Deploy on Save

If selected, files are compiled and deployed when you save them. This option saves you time when you run or debug your application in the IDE.

VM Options:
(used for running main classes or unit tests; e.g. -Xms10m)

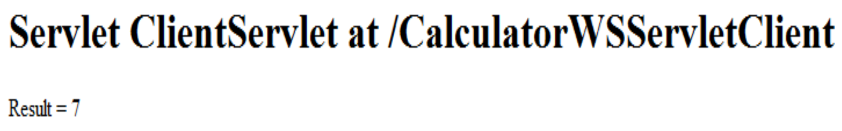
- In the Source Editor, remove the line that comments out the body of the processRequest method;

- Drag the node that represents the add operation into the space that you created;



```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet ClientServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet ClientServlet at " + request.getContextPath () + "</h1>");

    try { // Call Web Service Operation
        org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
        // TODO initialize WS operation arguments here
        int i = 3;
        int j = 4;
        // TODO process result here
        int result = port.add(i, j);
        out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }
} finally {
    out.println("</body>");
    out.println("</html>");
    out.close();
}
```

- Right-click the project node and choose Run;



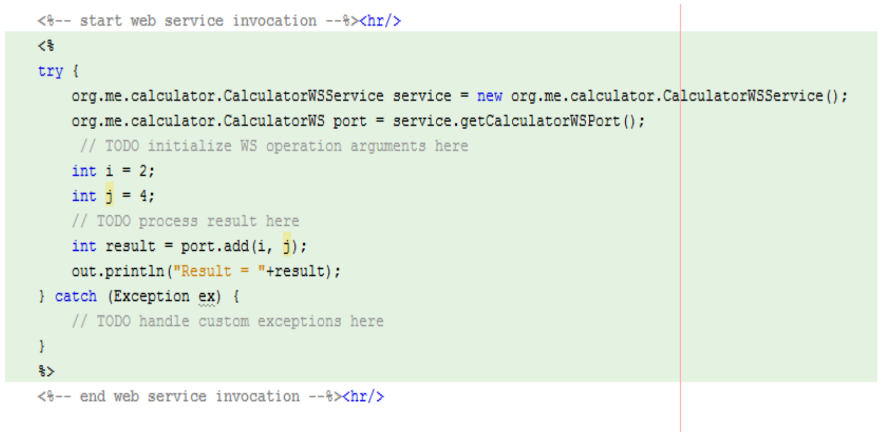
Servlet ClientServlet at /CalculatorWSServletClient

Result = 7

Client 3: JSP Page in Web Application

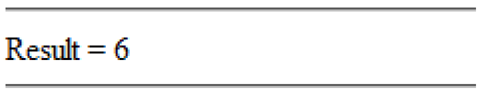
- Choose File -> New Project (Ctrl-Shift-N). Select Web Application from the Java Web category. Name the project CalculatorWSJSPClient. Click Finish;
- Right-click the CalculatorWSJSPClient node and choose New -> Web Service Client;

- In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK;
- Leave the other settings at default and click Finish;
- In the Web Service References node, expand the node that represents the web service. The add operation, which you will invoke from the client, is now exposed;
- Drag the add operation to the client's index.jsp page, and drop it below the H1 tags. The code for invoking the service's operation is now generated in the index.jsp page, a



```
<%-- start web service invocation --%><hr/>
<%
try {
    org.me.calculator.CalculatorWSService service = new org.me.calculator.CalculatorWSService();
    org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
     // TODO initialize WS operation arguments here
    int i = 2;
    int j = 4;
    // TODO process result here
    int result = port.add(i, j);
    out.println("Result = "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
<%-- end web service invocation --%><hr/>
```

- Right-click the project node and choose Run;



```
Result = 6
```

- http://netbeans.org/kb/docs/websvc/jax-ws.html

3. Class work

Ex1: Test the examples presented in chapter 2.

Ex2: Create a simple Web Service that converts the temperature from Fahrenheit to Celsius, and vice versa. Create 3 different clients to make use of the Web service.

Fahrenheit to Celsius:

$$C = (F - 32) \cdot \frac{5}{9}$$

Celsius to Fahrenheit:

$$F = C \cdot \frac{9}{5} + 32$$

4. Home Work

Ex1: Make a presentation about a Real-life Web Services .

- http://www.javapassion.com/webservices/#Real-life_Web_Services__SOA

Lesson 7

C# Web Services

1. Introduction

A Web service is a class that allows its methods to be called by methods on other machines via common data formats and protocols, such as XML and HTTP. In .NET, the over-the-network method calls are commonly implemented through the Simple Object Access Protocol (SOAP), an XML-based protocol describing how to mark up requests and responses so that they can be transferred via protocols such as HTTP. Using SOAP, applications represent and transmit data in a standardized XML-based format.

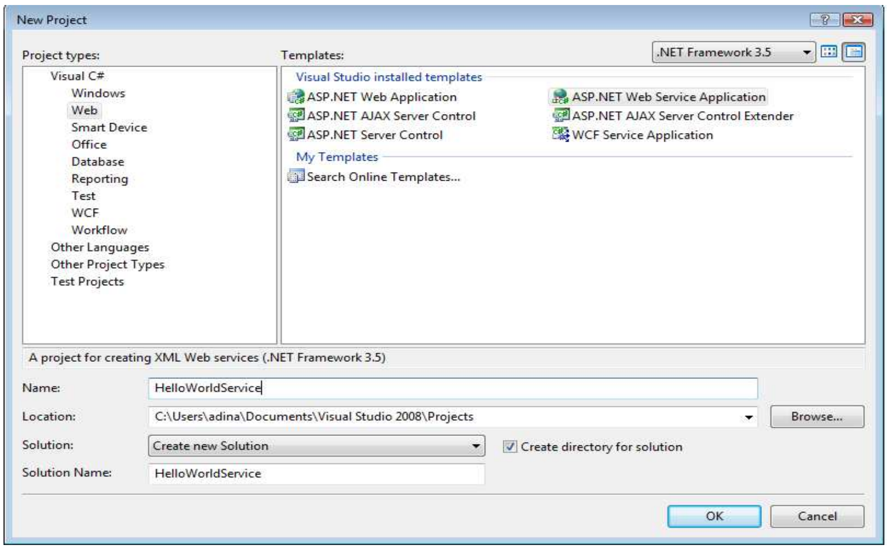
Web services have important implications for business-to-business (B2B) transactions. They enable businesses to conduct transactions via standardized, widely available Web services rather than relying on proprietary applications. Web services and SOAP are platform and language independent, so companies can collaborate via Web services without worrying about the compatibility of their hardware, software and communications technologies. Companies such as Amazon, Google, eBay and many others are using Web services to their advantage. To read case studies of Web services used in business, visit msdn.microsoft.com/webservices/understanding/casestudies/default.aspx.

Visual Web Developer and the .NET Framework provide a simple, user-friendly way to create Web services. In this paper, we learn how to use these tools to create, deploy and use Web services.

2. Web service applications using the wizards in Visual Studio. NET

Creating your first web service using the wizards in Visual Studio. NET is incredibly easy. You can have your first service up and running in minutes with no coding.

To create a new web service, fire up Visual Studio .NET and either select the New Project button on the default Start Page or click File →New →Project on the main Visual Studio .NET menu bar. Here you have the option to create a variety of project types from templates. Under Visual C# Projects, one template option creates an ASP.NET web service.



A project for creating XML Web services (.NET Framework 3.5)

Name: HelloWorldService

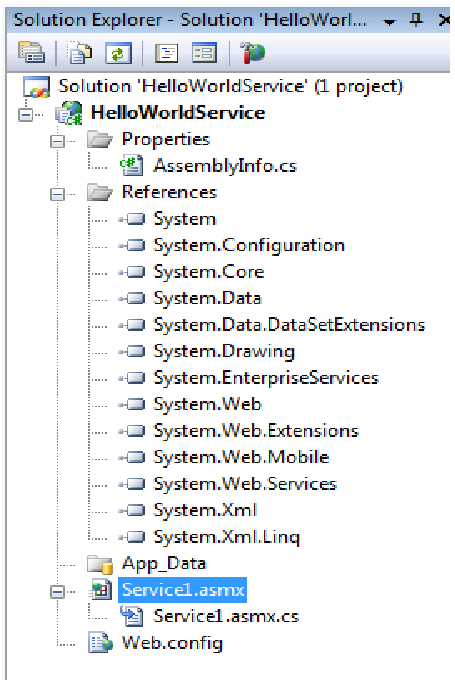
Location: C:\Users\adina\Documents\Visual Studio 2008\Projects

Solution: Create new Solution ☑ Create directory for solution

Solution Name: HelloWorldService

Once you click OK, the IDE (Integrated Development Environment) creates a new solution project and automatically populate the project with several files. The IDE will also create a virtual folder with the same name as the project name, which, in this case, is HelloWorldService.

The contents of your new project are displayed in the Solution Explorer window, which should appear on the right side of the VS.NET IDE.



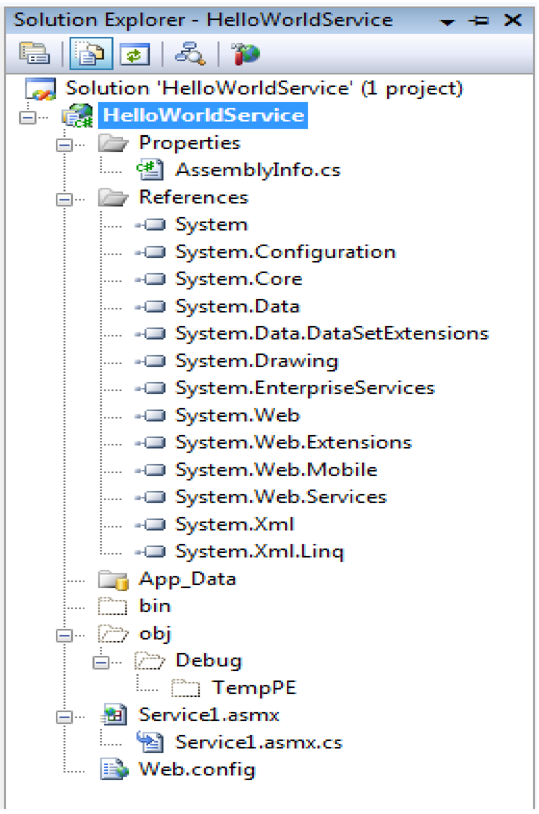
Solution Explorer - Solution 'HelloWorl...

```
Solution 'HelloWorldService' (1 project)
  HelloWorldService
    Properties
      AssemblyInfo.cs
    References
      System
      System.Configuration
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Drawing
      System.EnterpriseServices
      System.Web
      System.Web.Extensions
      System.Web.Mobile
      System.Web.Services
      System.Xml
      System.Xml.Linq
    App_Data
    Service1.asmx
      Service1.asmx.cs
    Web.config
```

If the Solution Explorer is not visible, you can open it by selecting Solution Explorer from the View menu.

VS.NET has included assembly references to the System, System.Data, System.Web, System.Web.Services, and System.XML namespaces. (The System.Data and System.XML assembly references are not necessary for this example, so you can remove them if you'd like, but there's no real benefit to doing so other than simplicity.)

To see all files, select Show All Files from the Project menu tab (there's also an icon at the top of Solution Explorer to do this). The Solution Explorer view will change to look.



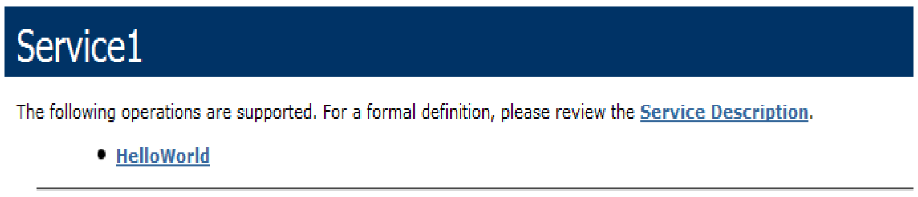
- AssemblyInfo.cs - An information file that provides the compiler with metadata (name, version, etc.) about the assemblies in the project;
- Web.config - An XML file containing configuration information for the application;
- *Service1.asmx.cs* - code view page;

This boilerplate code begins by importing several namespaces generally required for web services and by automatically generating namespace and class definitions. In this example, the namespace and class definitions are HelloWorldService and Service1, respectively.

The namespace definition is generated based on the project name, but you will probably want to change to something more depending on your application. The service name is always autogenerated as *Service1*. Change this to something more appropriate for your application, but you should also remember to change the name of the *.asmx* page to mirror your service name. Your service will run just fine if the names don't match up, but keeping the naming consistent can help make managing your service easier, particularly if you have a project with a large number of services.

The imported namespaces at the beginning of the code are provided as a convenience, and some of them are unnecessary. Specifically, the System.Data, System.Collections namespaces are not used at all. The classes of the System.ComponentModel namespace are used only by the web service designer methods, InitializeComponent() and Dispose(), which work in conjunction with a private member variable of type IContainer called components. To see these methods, you need to expand the Component Designer Generated Code region.

Once your application is complete, select Build Solution from the Build menu (or press Ctrl-Shift-B) and VS.NET will compile your web service and transfer the *.asmx* page and associated compiled assembly to the web server for you. If any errors result from the compile, VS.NET will display them in a panel labeled Output at the bottom of the IDE. Once you have successfully built the web service, it's ready to be used.



Service1

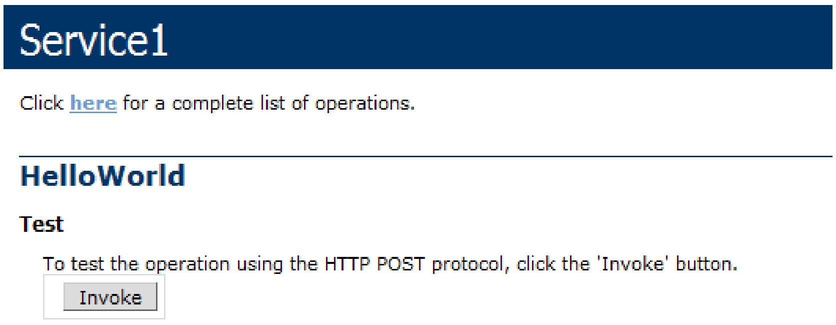
The following operations are supported. For a formal definition, please review the Service Description.

- HelloWorld

If you mouse over the HelloWorld link, you'll see the destination URL:

http://localhost/HelloWorldService.cs.asmx?op=HelloWorld

By clicking this link, you call the *.asmx* page, passing a parameter called `op` (standing presumably for operation) along with the name of the service. This action is the same as calling the `HelloWorld` web method of the web service using HTTP GET.


Service1

Click here for a complete list of operations.

HelloWorld

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Here you'll see the name of the service and method along with a button to test the service. Through reflection, the logic in the *DefaultWsdlHelpGenerator.aspx* test page is able to determine the signature of our HelloWorld method. Because our web method takes no arguments, the page need provide only a button for invocation. If the method had a different signature, for example, if it reads a string of text, the *.aspx* help page would also provide a text box to capture this string and pass it, using HTTP GET, to the web method when the form was submitted. This text box method works fine for simple data type arguments, but if the web method were to require an object, this approach would not work.

You can invoke the web method using the IE test page by opening a web browser and navigating to the service's URL. You will see a page listing the service's operation, which should be HelloWorld. Click the HelloWorld operation to navigate to the web method invocation page. This is a page that allows you to test the operation by clicking a button labeled Invoke. To invoke the service, click the button.

The deployment process in Visual Studio .NET is as simple as choosing the Build Solution option from the Build menu item (or pressing Ctrl-Shift-B). But, in order to take advantage of this two-click deployment, you first need to properly configure Visual Studio .NET to be able to deploy to your instance of IIS.

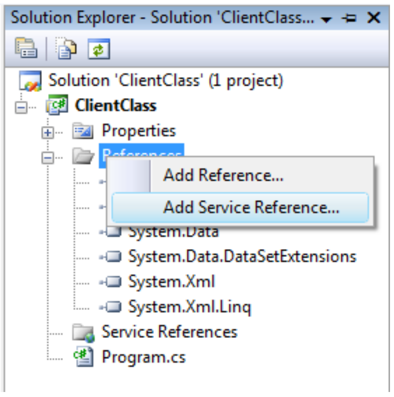
This service is only returning a string. Strings as well as most numbers are considered simple types. Web services are not limited to these simple types for return values or inbound parameters. The next web service will demonstrate how to return a complex type. For this create a web service that returns the date and time from the server. Instead, create a LocalTime struct that will be returned to the caller. The LocalTime struct holds seven values; Day, Month, Year, Hour, Minute, Seconds, Milliseconds, and Timezone. The struct's definition is below:

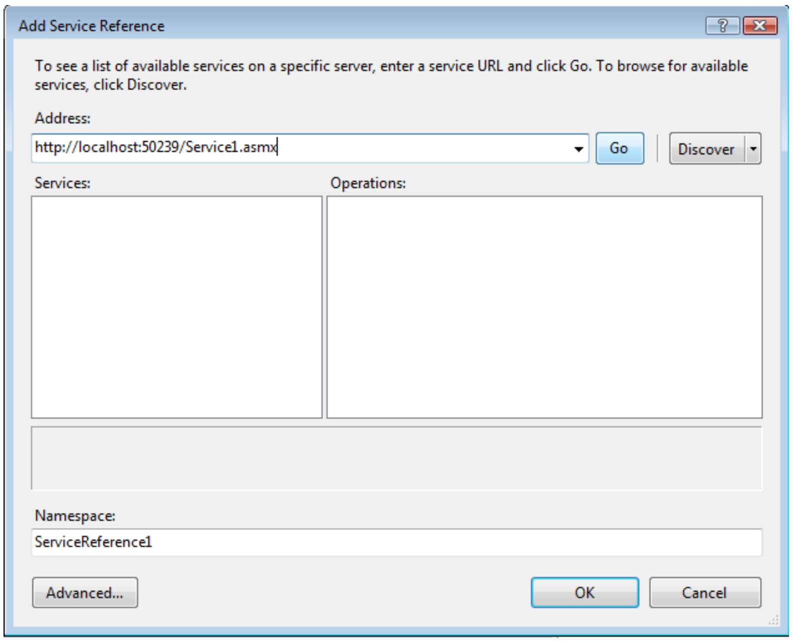
```
public struct LocalTime
{
public int Day;
public int Month;
public int Year;
public int Hour;
public int Minute;
public int Seconds;
public int Milliseconds;
public string Timezone;
}


public class TimeService
{
[WebMethod]
public LocalTime GetTime()
{
LocalTime lt = new LocalTime();
DateTime dt = DateTime.Now;
lt.Day = dt.Day;
lt.Month = dt.Month;
lt.Year = dt.Year;
lt.Hour = dt.Hour;
lt.Minute = dt.Minute;
lt.Seconds = dt.Second;
lt.Milliseconds = dt.Millisecond;
lt.Timezone = TimeZone.CurrentTimeZone.StandardName;
return lt;
}
}
```

To demonstrate that the time web service is usable by any .Net application, create a simple console application in C# that prints out the time from the remote service.

```
using System;
class ClientClass
{
static void Main()
{
TimeService ts = new TimeService();
LocalTime lt = ts.GetTime();
string stime = lt.Hour + ":" + lt.Minute + ":" + lt.Seconds + "." +
lt.Milliseconds + " " + lt.Timezone;
string sdate = lt.Month + "/" + lt.Day + "/" + lt.Year;
Console.WriteLine("The remote date is: " + sdate);
Console.WriteLine("The remote time is: " + stime);
}
}
```

After creating the client application, add a proxy class to the project that allows the client to access the Web service. Recall that the proxy class (or proxy) is generated from the Web service's WSDL file and enables the client to call Web methods over the Internet. The proxy class handles all the details of communicating with the Web service. The proxy class is hidden from you by default-you can view it in the Solution Explorer by clicking the Show All Files button. The proxy class's purpose is to make clients think that they are calling the Web methods directly. The next example demonstrates how to create a Web service client and generate a proxy class that allows the client to access the Web service. Begin by adding a Web reference to the client project. When you add the Web reference, Visual C# will generate the appropriate proxy class. You will then create an instance of the proxy class and use it to call the Web service's methods. To add Web reference perform the following steps:




Add Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:50239/Service1.asmx

Namespace: ServiceReference1

3. Class work

Ex1: Test the examples presented in chapter 2.

Ex2: Create a simple Web Service that converts the temperature from Fahrenheit to Celsius, and vice versa. Create 2 different clients to make use of the Web service.

Fahrenheit to Celsius:

$$C = (F - 32) \cdot \frac{5}{9}$$

Celsius to Fahrenheit:

$$F = C \cdot \frac{9}{5} + 32$$

4. Home Work

Create a Web Service that give access to a Data Base where are store touristic information's about Cluj-Napoca City. Create a client to make use of the Web service.