**William Collins**

2/9/2012

**IPv4 Calculator**

**Assignment 1**

**Coll0300 – 040652633 – coll0300@algonquinlive.com**

**CST 8218 – 310 – Web Application Programming**

**For: Gerald Hurdle**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* \*\*/

/\*\* \*\*/

/\*\* Student Name : William Collins \*\*/

/\*\* EMail Address : coll0300@algonquinlive.com \*\*/

/\*\* Student Number : 040652633 \*\*/

/\*\* Student User ID : coll0300 \*\*/

/\*\* Course Number : CST 8218 \*\*/

/\*\* Lab Section Number : 310 \*\*/

/\*\* Professor Name : Gerald Hurdle \*\*/

/\*\* Assignment Name/Number/Date : IP Calculator / 1 / 2/7/2012\*\*/

/\*\* Optional Comments : \*\*/

/\*\* \*\*/

/\*\* \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

package controller;

import java.util.ArrayList;

import java.util.Collection;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

import model.IpCalculatorModel;

/\*\*

\*

\* @author willy

\*/

@ManagedBean(name="controller")

@SessionScoped

public class IpCalculatorController {

private IpCalculatorModel ip;

private String baseAddressString;

private String baseAddressBinary;

private String maskString;

private String maskBinary;

private String broadcastAddressString;

private String broadcastAddressBinary;

private String addressClass;

private int numberOfHostBits;

private int numberOfHosts;

private boolean isPrivateAddress;

private boolean isDefaultRoute;

private boolean isLoopback;

private boolean isLinkLocal;

private boolean isTestNet;

private String subnetMaskString;

private String subnetMaskBinary;

private String wildcardMaskString;

private String wildcardMaskBinary;

private int subnetBits;

private int useableSubnets;

private int hostBitsPerSubnet;

private int usableHostsPerSubnet;

private String subnetAddressString;

private String subnetAddressBinary;

private String firstHostString;

private String firstHostBinary;

private String lastHostString;

private String lastHostBinary;

private String subnetBroadcastAddressString;

private String subnetBroadcastAddressBinary;

private Collection prefixValues;

public IpCalculatorController() {

super();

ip = new IpCalculatorModel();

calculateIpInformation();

//Set up the collection that is used in the drop down menu

prefixValues = new ArrayList();

for (int i = 0; i < 31; i++){

prefixValues.add(i);

}

}

public String getAddressClass() {

return addressClass;

}

public String getBaseAddressBinary() {

return baseAddressBinary;

}

public String getBaseAddressString() {

return baseAddressString;

}

public String getBroadcastAddressBinary() {

return broadcastAddressBinary;

}

public String getBroadcastAddressString() {

return broadcastAddressString;

}

public String getFirstHostBinary() {

return firstHostBinary;

}

public String getFirstHostString() {

return firstHostString;

}

public int getHostBitsPerSubnet() {

return hostBitsPerSubnet;

}

public boolean isIsDefaultRoute() {

return isDefaultRoute;

}

public boolean isIsLinkLocal() {

return isLinkLocal;

}

public boolean isIsLoopback() {

return isLoopback;

}

public boolean isIsPrivateAddress() {

return isPrivateAddress;

}

public boolean isIsTestNet() {

return isTestNet;

}

public IpCalculatorModel getIp() {

return ip;

}

public String getLastHostBinary() {

return lastHostBinary;

}

public String getLastHostString() {

return lastHostString;

}

public String getMaskBinary() {

return maskBinary;

}

public String getMaskString() {

return maskString;

}

public int getNumberOfHosts() {

return numberOfHosts;

}

public int getNumberOfHostBits() {

return numberOfHostBits;

}

public Collection getPrefixValues() {

return prefixValues;

}

public String getSubnetAddressBinary() {

return subnetAddressBinary;

}

public String getSubnetAddressString() {

return subnetAddressString;

}

public int getSubnetBits() {

return subnetBits;

}

public String getSubnetMaskBinary() {

return subnetMaskBinary;

}

public String getSubnetMaskString() {

return subnetMaskString;

}

public int getUsableHostsPerSubnet() {

return usableHostsPerSubnet;

}

public int getUseableSubnets() {

return useableSubnets;

}

public String getWildcardMaskBinary() {

return wildcardMaskBinary;

}

public String getWildcardMaskString() {

return wildcardMaskString;

}

public String getSubnetBroadcastAddressBinary() {

return subnetBroadcastAddressBinary;

}

public String getSubnetBroadcastAddressString() {

return subnetBroadcastAddressString;

}

public final String calculateIpInformation(){

//Make sure that the ip address is compressed and ready for calculations

ip.calculate();

//Perform IP Calculations and assign values to local properties

baseAddressString = formatIpString(intToIntArray(ip.getBaseAddress()));

baseAddressBinary = formatIpStringBinary(intToIntArray(ip.getBaseAddress()));

maskString = formatIpString(intToIntArray(ip.getMaskAddress()));

maskBinary = formatIpStringBinary(intToIntArray(ip.getMaskAddress()));

broadcastAddressString = formatIpString(intToIntArray(ip.getBroadcastAddress()));

broadcastAddressBinary = formatIpStringBinary(intToIntArray(ip.getBroadcastAddress()));

addressClass = ip.getNetworkClass();

numberOfHostBits = ip.getNumberOfHostBits();

numberOfHosts = ip.getNumberOfUsableHosts();

isPrivateAddress = ip.isPrivateAddress();

isDefaultRoute = ip.isDefaultRoute();

isLoopback = ip.isLoopback();

isLinkLocal = ip.isLinkLocal();

isTestNet = ip.isTestNet();

subnetMaskString = formatIpString(intToIntArray(ip.getSubnetMask()));

subnetMaskBinary = formatIpStringBinary(intToIntArray(ip.getSubnetMask()));

wildcardMaskString = formatIpString(intToIntArray(ip.getWildcardMask()));

wildcardMaskBinary = formatIpStringBinary(intToIntArray(ip.getWildcardMask()));

subnetBits = ip.getNumberOfSubnetBits();

useableSubnets = ip.getNumberOfUsableSubnets();

hostBitsPerSubnet = ip.getNumberOfHostBitsPerSubnet();

usableHostsPerSubnet = ip.getNumberOfHostsPerSubnet();

subnetAddressString = formatIpString(intToIntArray(ip.getSubnetAddress()));

subnetAddressBinary = formatIpStringBinary(intToIntArray(ip.getSubnetAddress()));

firstHostString = formatIpString(intToIntArray(ip.getFirstHost()));

firstHostBinary = formatIpStringBinary(intToIntArray(ip.getFirstHost()));

lastHostString = formatIpString(intToIntArray(ip.getLastHost()));

lastHostBinary = formatIpStringBinary(intToIntArray(ip.getLastHost()));

subnetBroadcastAddressString = formatIpString(intToIntArray(ip.getSubnetBroadcastAddress()));

subnetBroadcastAddressBinary = formatIpStringBinary(intToIntArray(ip.getSubnetBroadcastAddress()));

return "index";

}

/\*\*

\* Creates a traditional four octet IP string out of an array of 4 integers

\* representing the four octets of the IP address

\*

\* @param ip An array of 4 IP address octets as integers

\* @return The string formatted IP address

\*/

private String formatIpString(int[] ip) {

StringBuilder builder = new StringBuilder();

for (int i = 0; i < 4; i++) {

builder.append(ip[i]).append('.');

}

builder.deleteCharAt(builder.length() - 1);

return builder.toString();

}

/\*\*

\* Creates a string of the binary view of an IP address.

\*

\* @param ip An array of four integers representing an IP Address

\* @return The formatted representation of the binary IP adddress

\*/

private String formatIpStringBinary(int[] ip) {

StringBuilder builder = new StringBuilder();

for (int b : ip) {

for (int i = 128; i > 0; i /= 2) {

if ((b & i) > 0) {

builder.append('1');

} else {

builder.append('0');

}

}

builder.append('.');

}

builder.deleteCharAt(builder.length() - 1);

return builder.toString();

}

/\*\*

\* Breaks up the four bytes of an integer and returns an array of each octet

\* in integer format. Integers are easier to work with than bytes.

\*

\* @param number The packed integer representing an IP address

\* @return The broken array of integers containing each IP octet

\*/

private int[] intToIntArray(int number) {

int[] newInt = new int[4];

for (int i = 3; i >= 0; i--) {

newInt[i] = (number >>> 8 \* (3 - i)) & (0xFF);

}

return newInt;

}

@Override

public String toString(){

return "IP Calculator Controller";

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* \*\*/

/\*\* \*\*/

/\*\* Student Name : William Collins \*\*/

/\*\* EMail Address : coll0300@algonquinlive.com \*\*/

/\*\* Student Number : 040652633 \*\*/

/\*\* Student User ID : coll0300 \*\*/

/\*\* Course Number : CST 8218 \*\*/

/\*\* Lab Section Number : 310 \*\*/

/\*\* Professor Name : Gerald Hurdle \*\*/

/\*\* Assignment Name/Number/Date : IP Calculator / 1 / 2/9/2012\*\*/

/\*\* Optional Comments : \*\*/

/\*\* \*\*/

/\*\* \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

package model;

import java.io.Serializable;

/\*\*

\*

\* @author willy

\*/

public class IpCalculatorModel implements Serializable {

/\*

\* CLASS VALUES

\*/

private static final int CLASS\_A\_MASK = 0x80000000;

private static final int CLASS\_B\_MASK = 0xC0000000;

private static final int CLASS\_C\_MASK = 0xC0000000;

private static final int CLASS\_A = 0x00000000;

private static final int CLASS\_B = 0x80000000;

private static final int CLASS\_C = 0xC0000000;

/\*

\* MASKS - masks used for determining certain characteristics of an adress

\*/

private static final int DEFAULT\_ROUTE\_MASK = 0xFF000000;

private static final int LOOPBACK\_MASK = 0xFF000000;

private static final int LINK\_LOCAL\_MASK = 0xFFFF0000;

private static final int TEST\_NET\_MASK = 0xFFFFFF00;

private static final int[] PRIVATE\_ADDRESS\_MASKS = {

0xFF000000,

0xFFF00000,

0xFFFF0000

};

/\*

\* ADDRESS BLOCKS - The IP blocks of certain known address types

\*/

private static final int DEFAULT\_ROUTE = 0x00000000;

private static final int LOOPBACK = 0x7F000000;

private static final int LINK\_LOCAL = 0xA9FE0000;

private static final int TEST\_NET = 0xC0000200;

private static final int[] PRIVATE\_ADDRESSES = {

0x0A000000, //10.0.0.0 - 10.255.255.255

0xAC100000, //172.16.0.0 - 172.31.255.255

0xC0A80000 //192.168.0.0 - 192.168.255.255

};

//The octets are bean properties represting the octets in an IP address

private int octet1;

private int octet2;

private int octet3;

private int octet4;

private int address; //The IP address

private int mask; //The mask representation of the network prefix

private int broadcast; //The broadcast address of the subnet

private int network; //The network address of the subnet

private int classNetworkMask; //Mask of the class type of the network.

public IpCalculatorModel() {

//A nice set of default values for the octet and prefix fields

octet1 = 192;

octet2 = 168;

octet3 = 0;

octet4 = 1;

mask = 0xFFFFFF00;

}

/\*\*

\* Retrieves the subnet prefix from the local bit mask for the address

\*

\* @return An integer value representing a subnet prefix

\*/

public int getMask() {

int tally = 0;

//tally the number of bits from the left hand side of the mask

for (int i = 0; i < 32; i++) {

tally += ((mask << i) & 0xEFFFFFFF) >>> 31;

}

return tally;

}

/\*\*

\* Takes an integer representing a prefix, and converts it to a compress

\* integer bitmask.

\*

\* @param prefix The subnet prefix representing the bit mask

\*/

public void setMask(int prefix) {

int localmask = 0;

for (int i = 0; i < prefix; i++) {

localmask |= (0x80000000 >>> i);

}

this.mask = localmask;

}

public int getOctet1() {

return octet1;

}

public void setOctet1(int octet1) {

this.octet1 = octet1;

}

public int getOctet2() {

return octet2;

}

public void setOctet2(int octet2) {

this.octet2 = octet2;

}

public int getOctet3() {

return octet3;

}

public void setOctet3(int octet3) {

this.octet3 = octet3;

}

public int getOctet4() {

return octet4;

}

public void setOctet4(int octet4) {

this.octet4 = octet4;

}

/\*\*

\* \*\*\*\*\*\*\*\*\*CALCULATION METHODS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

public int getBaseAddress() {

return (address & classNetworkMask);

}

public int getSubnetAddress() {

return network;

}

public int getMaskAddress() {

return classNetworkMask;

}

public int getSubnetMask(){

return mask;

}

public int getBroadcastAddress(){

return address | (~classNetworkMask);

}

public int getSubnetBroadcastAddress() {

return broadcast;

}

public String getNetworkClass() {

if ( (address & CLASS\_A\_MASK) == CLASS\_A) {

return "A";

}

if ( (address & CLASS\_B\_MASK) == CLASS\_B) {

return "B";

}

if ( (address & CLASS\_C\_MASK) == CLASS\_C) {

return "C";

}

return "Unknown";

}

public int getNumberOfHostBitsPerSubnet() {

return calculateNumberOfHostBits(mask);

}

public int getNumberOfHostBits() {

return calculateNumberOfHostBits(classNetworkMask);

}

public int getNumberOfUsableHosts() {

return (int) Math.pow(2.0, (double) getNumberOfHostBits()) - 2;

}

public int getNumberOfHostsPerSubnet() {

return (int) Math.pow(2.0, (double) getNumberOfHostBitsPerSubnet()) - 2;

}

public int getNumberOfSubnetBits() {

int subnetBits = getNumberOfHostBits() - getNumberOfHostBitsPerSubnet();

if (subnetBits < 0)

subnetBits = 0;

return subnetBits;

}

public int getNumberOfUsableSubnets() {

int subnetBits = getNumberOfSubnetBits();

if (subnetBits == 0)

return 0;

return (int) Math.pow(2, getNumberOfSubnetBits());

}

public int getWildcardMask() {

return ~mask;

}

public int getFirstHost() {

return network + 1;

}

public int getLastHost() {

return broadcast - 1;

}

public boolean isPrivateAddress() {

for (int i = 0; i < 3; i++) {

if ((address & PRIVATE\_ADDRESS\_MASKS[i]) == PRIVATE\_ADDRESSES[i])

return true;

}

return false;

}

public boolean isDefaultRoute() {

if ((address & DEFAULT\_ROUTE\_MASK) == DEFAULT\_ROUTE)

return true;

return false;

}

public boolean isLoopback() {

if ((address & LOOPBACK\_MASK) == LOOPBACK)

return true;

return false;

}

public boolean isLinkLocal() {

if ((address & LINK\_LOCAL\_MASK) == LINK\_LOCAL)

return true;

return false;

}

public boolean isTestNet() {

if ((address & TEST\_NET\_MASK) == TEST\_NET)

return true;

return false;

}

/\*\*

\* This method should only be invoked after reading in all four IP octets.

\* It will pack the octets into a single integer variable in order to

\* ease the process of manipulating and analyzing the bits inside.

\*/

public void calculate() {

compressOctetsToInt();

calculateSubnetBroadcastAddress();

calculateNetworkAddress();

calculateAddressClass();

}

private void calculateSubnetBroadcastAddress() {

broadcast = (~mask) | address;

}

private void calculateNetworkAddress() {

network = address & mask;

}

private void calculateAddressClass() {

if ((address & CLASS\_A\_MASK) == CLASS\_A) {

classNetworkMask = 0xFF000000;

}

else if ((address & CLASS\_B\_MASK) == CLASS\_B) {

classNetworkMask = 0xFFFF0000;

}

else if ((address & CLASS\_C\_MASK) == CLASS\_C) {

classNetworkMask = 0xFFFFFF00;

}

else

classNetworkMask = 0x00000000;

}

/\*\*

\* Calculates the number of host bits given a network or subnet mask.

\* @param mask The network or subnet mask to be analyzed

\* @return The number of host bits in the mask

\*/

private int calculateNumberOfHostBits(int mask){

int tally = 0;

while ((~mask >>> tally) != 0) {

tally++;

if (tally > 32) return 32;

}

return tally;

}

/\*\*

\* Takes the octet properties that are mapped to the jsf page and packs them

\* into a single integer names "address". This allows for easy bitwise

\* operations using the mask.

\*/

private void compressOctetsToInt() {

address = 0;

address |= octet1 << 24;

address |= octet2 << 16;

address |= octet3 << 8;

address |= octet4;

}

@Override

public String toString() {

return "IP Calculator Model";

}

}

<?xml version='1.0' encoding='UTF-8' ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:ui="http://java.sun.com/jsf/facelets"

xmlns:h="http://java.sun.com/jsf/html"

xmlns:f="http://java.sun.com/jsf/core">

<body>

<ui:composition template="./WEB-INF/templates/template.xhtml">

<ui:define name="content">

<h:form id="calculatorForm" >

<!--\*\*\*\*\*\*\*\*\*\*\*\*-->

<!-- INPUT AREA -->

<!--\*\*\*\*\*\*\*\*\*\*\*\*-->

<h:panelGrid id="addressPanel" border="1" columns="2">

<h:panelGroup>

<div class="title">

<h:outputText id="ipAddress"

value="#{msg.ipAddress}"/>

</div>

<h:inputText id="octet1"

label="Octet 1"

class="ipoctet"

required="true"

requiredMessage="Octet 1 is required."

maxlength="3"

value="#{controller.ip.octet1}">

<f:validateLongRange minimum="0" maximum="223"/>

</h:inputText>

<h:inputText id="octet2"

label="Octet 2"

class="ipoctet"

required="true"

requiredMessage="Octet 2 is required."

maxlength="3"

value="#{controller.ip.octet2}">

<f:validateLongRange minimum="0" maximum="255"/>

</h:inputText>

<h:inputText id="octet3"

label="Octet 3"

class="ipoctet"

required="true"

requiredMessage="Octet 3 is required."

maxlength="3"

value="#{controller.ip.octet3}">

<f:validateLongRange minimum="0" maximum="255"/>

</h:inputText>

<h:inputText id="octet4"

label="Octet 4"

class="ipoctet"

required="true"

requiredMessage="Octet 4 is required."

maxlength="3"

value="#{controller.ip.octet4}">

<f:validateLongRange minimum="0" maximum="255"/>

</h:inputText>

</h:panelGroup>

<h:panelGroup>

<div class="title">

<h:outputText id="prefix"

value="#{msg.prefix}"/>

</div>

<h:selectOneMenu value="#{controller.ip.mask}">

<f:selectItems value="#{controller.prefixValues}"/>

</h:selectOneMenu>

</h:panelGroup>

</h:panelGrid>

<p/>

<h:commandButton id="calculateButton"

value="#{msg.calculate}"

action="#{controller.calculateIpInformation()}"/>

<h:messages errorClass="errors"/>

<hr/>

</h:form>

<!--\*\*\*\*\*\*\*\*\*\*\*\*-->

<!-- OUTPUT AREA-->

<!--\*\*\*\*\*\*\*\*\*\*\*\*-->

<p/>

<h:panelGrid columns="2">

<h:panelGrid id="majorPanel" border="1" columns="3">

<h:outputText value="#{msg.baseAddress}"/>

<h:outputText value="#{controller.baseAddressString}"/>

<h:outputText value="#{controller.baseAddressBinary}"/>

<h:outputText value="#{msg.mask}"/>

<h:outputText value="#{controller.maskString}"/>

<h:outputText value="#{controller.maskBinary}"/>

<h:outputText value="#{msg.broadcastAddress}"/>

<h:outputText value="#{controller.broadcastAddressString}"/>

<h:outputText value="#{controller.broadcastAddressBinary}"/>

</h:panelGrid>

<h:panelGrid id="classPanel" border="1" columns="2">

<h:outputText value="#{msg.class}"/>

<h:outputText value="#{controller.addressClass}"/>

<h:outputText value="#{msg.hostBits}"/>

<h:outputText value="#{controller.numberOfHostBits}"/>

<h:outputText value="#{msg.usableHosts}"/>

<h:outputText value="#{controller.numberOfHosts}"/>

</h:panelGrid>

</h:panelGrid>

<!--CHECKBOXES-->

<p/>

<h:panelGrid id="ipAttributePanel" border="1" columns="5">

<h:outputLabel value="#{msg.privateAddress}">

<h:selectBooleanCheckbox disabled="true"

value="#{controller.isPrivateAddress}"/>

</h:outputLabel>

<h:outputLabel value="#{msg.defaultRoute}">

<h:selectBooleanCheckbox disabled="true"

value="#{controller.isDefaultRoute}"/>

</h:outputLabel>

<h:outputLabel value="#{msg.loopback}">

<h:selectBooleanCheckbox disabled="true"

value="#{controller.isLoopback}"/>

</h:outputLabel>

<h:outputLabel value="#{msg.linkLocal}">

<h:selectBooleanCheckbox disabled="true"

value="#{controller.isLinkLocal}"/>

</h:outputLabel>

<h:outputLabel value="#{msg.testNet}">

<h:selectBooleanCheckbox disabled="true"

value="#{controller.isTestNet}"/>

</h:outputLabel>

</h:panelGrid>

<!--SUBNET INFORMATION-->

<p/>

<h:panelGrid id="subnetPanel" columns="3">

<h:panelGrid border="1" columns="3">

<h:outputText value="#{msg.subnetMask}"/>

<h:outputText value="#{controller.subnetMaskString}"/>

<h:outputText value="#{controller.subnetMaskBinary}"/>

<h:outputText value="#{msg.wildcardMask}"/>

<h:outputText value="#{controller.wildcardMaskString}"/>

<h:outputText value="#{controller.wildcardMaskBinary}"/>

</h:panelGrid>

<h:panelGrid border="1" columns="2">

<h:outputText value="#{msg.subnetBits}"/>

<h:outputText value="#{controller.subnetBits}"/>

<h:outputText value="#{msg.usableSubnets}"/>

<h:outputText value="#{controller.useableSubnets}"/>

</h:panelGrid>

<h:panelGrid border="1" columns="2">

<h:outputText value="#{msg.hostBitsPerSubnet}"/>

<h:outputText value="#{controller.hostBitsPerSubnet}"/>

<h:outputText value="#{msg.hostsPerSubnet}"/>

<h:outputText value="#{controller.usableHostsPerSubnet}"/>

</h:panelGrid>

</h:panelGrid>

<!--ADDRESS INFORMATION-->

<p/>

<h:panelGrid id="addressInfoPanel" border="1" columns="3">

<h:outputText value="#{msg.subnetForThisIp}"/>

<h:outputText value="#{controller.subnetAddressString}"/>

<h:outputText value="#{controller.subnetAddressBinary}"/>

<h:outputText value="#{msg.firstHost}"/>

<h:outputText value="#{controller.firstHostString}"/>

<h:outputText value="#{controller.firstHostBinary}"/>

<h:outputText value="#{msg.lastHost}"/>

<h:outputText value="#{controller.lastHostString}"/>

<h:outputText value="#{controller.lastHostBinary}"/>

<h:outputText value="#{msg.broadcastForThisSubnet}"/>

<h:outputText value="#{controller.subnetBroadcastAddressString}"/>

<h:outputText value="#{controller.subnetBroadcastAddressBinary}"/>

</h:panelGrid>

</ui:define>

</ui:composition>

</body>

</html>

#Messages for the IPv4 Calculator

ipAddress=Ip Address

prefix=Prefix Length

calculate=IP Calculate

baseAddress=Major (base) network address

mask=Major network mask

broadcastAddress=Major network broadcast address

class=Class

hostBits=Total number of host bits

usableHosts=Number of usable hosts

privateAddress=Private Address?

defaultRoute=Default Route?

loopback=Loopback?

linkLocal=Link-Local?

testNet=Test-Net?

subnetMask=Subnet mask

wildcardMask=Wildcard mask

subnetBits=Number of subnet bits

usableSubnets=Number of usable subnets

hostBitsPerSubnet=Number of host bits per subnet

hostsPerSubnet=Number of usable hosts per subnet

subnetForThisIp=Subnet for this IP address

firstHost=IP address of first host on this subnet

lastHost=IP address of last host on this subnet

broadcastForThisSubnet=Broadcast address for this subnet

# ABOUT

course=CST8218 - Web Application Programming (WAP)

courseLabel=Course

professor=Gerald.Hurdle@AlgonquinCollege.com

professorLabel=Professor

studentEmail=coll0300@algonquinlive.com

studentEmailLabel=Student Email

studentName=William Collins

studentNameLabel=Student Name

studentNumber=040 652 633

studentNumberLabel=Student Number

studentUsername=coll0300

studentUsernameLabel=Student Username

# FOOTER

semester=Winter

year=2012

<?xml version='1.0' encoding='UTF-8'?>

<!-- =========== FULL CONFIGURATION FILE ================================== -->

<faces-config version="2.0"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-facesconfig\_2\_0.xsd">

<application>

<locale-config>

<default-locale>en</default-locale>

<supported-locale>en</supported-locale>

</locale-config>

<resource-bundle>

<base-name>resources.about</base-name>

<var>about</var>

</resource-bundle>

<resource-bundle>

<base-name>resources.messages</base-name>

<var>msg</var>

</resource-bundle>

</application>

</faces-config>

/\*

Document : ipLayout

Created on : Feb 4, 2012, 2:04:12 PM

Author : William Collins

Description: Stylings associated with the ip input and output fields

\*/

.ipoctet {

width: 3em;

}

.title {

margin-left: auto;

margin-right: auto;

text-align: center;

font-weight: bold;

font-family: arial;

}

.errors {

color: red;

}

root {

display: block;

}