```
function isHexaDigit(digit) {
   var hexVals = new Array("0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
                           "A", "B", "C", "D", "E", "F", "a", "b", "c", "d", "e", "f");
   var len = hexVals.length;
   var i = 0;
   var ret = false;

   for ( i = 0; i < len; i++ )
      if ( digit == hexVals[i] ) break;

   if ( i < len )
      ret = true;

   return ret;
}

function isValidKey(val, size) {
   var ret = false;
   var len = val.length;
   var dbSize = size * 2;

   if ( len == size )
      ret = true;
   else if ( len == dbSize ) {
      for ( i = 0; i < dbSize; i++ )
         if ( isHexaDigit(val.charAt(i)) == false )
            break;
      if ( i == dbSize )
         ret = true;
   } else
      ret = false;

   return ret;
}


function isValidHexKey(val, size) {
   var ret = false;
   if (val.length == size) {
      for ( i = 0; i < val.length; i++ ) {
         if ( isHexaDigit(val.charAt(i)) == false ) {
            break;
         }
      }
      if ( i == val.length ) {
         ret = true;
      }
   }

   return ret;
}


function isNameUnsafe(compareChar) {
   var unsafeString = "\"<>%\\^[]`\+\$\,='#&@.: \t";

   if ( unsafeString.indexOf(compareChar) == -1 && compareChar.charCodeAt(0) > 32
```

```
        && compareChar.charCodeAt(0) < 123 )
      return false; // found no unsafe chars, return false
   else
      return true;
}

// Check if a name valid
function isValidName(name) {
   var i = 0;

   for ( i = 0; i < name.length; i++ ) {
      if ( isNameUnsafe(name.charAt(i)) == true )
         return false;
   }

   return true;
}

// same as is isNameUnsafe but allow spaces
function isCharUnsafe(compareChar) {
   var unsafeString = "\"<>%\\^[]`\+\$\,='#&@.:\t";

   if ( unsafeString.indexOf(compareChar) == -1 && compareChar.charCodeAt(0) >= 32
        && compareChar.charCodeAt(0) < 123 )
      return false; // found no unsafe chars, return false
   else
      return true;
}

function isValidNameWSpace(name) {
   var i = 0;

   for ( i = 0; i < name.length; i++ ) {
      if ( isCharUnsafe(name.charAt(i)) == true )
         return false;
   }

   return true;
}

function isSameSubNet(lan1Ip, lan1Mask, lan2Ip, lan2Mask) {

   var count = 0;

   lan1a = lan1Ip.split('.');
   lan1m = lan1Mask.split('.');
   lan2a = lan2Ip.split('.');
   lan2m = lan2Mask.split('.');

   for (i = 0; i < 4; i++) {
      l1a_n = parseInt(lan1a[i]);
      l1m_n = parseInt(lan1m[i]);
      l2a_n = parseInt(lan2a[i]);
      l2m_n = parseInt(lan2m[i]);
      if ((l1a_n & l1m_n) == (l2a_n & l2m_n))
         count++;
   }
```

```
   if (count == 4)
      return true;
   else
      return false;
}


function isValidIpAddress(address) {
   var i = 0;

   if ( address == '0.0.0.0' ||
        address == '255.255.255.255' )
      return false;

   addrParts = address.split('.');
   if ( addrParts.length != 4 ) return false;
   for (i = 0; i < 4; i++) {
      if (isNaN(addrParts[i]) || addrParts[i] =="")
         return false;
      num = parseInt(addrParts[i]);
      if ( num < 0 || num > 255 )
         return false;
   }
   return true;
}

function isValidIpAddress6(address) {
   var i = 0, num = 0;

   addrParts = address.split(':');
   if (addrParts.length < 3 || addrParts.length > 8)
      return false;
   for (i = 0; i < addrParts.length; i++) {
      if ( addrParts[i] != "" )
         num = parseInt(addrParts[i], 16);
      if ( i == 0 ) {
//         if ( (num & 0xf000) == 0xf000 )
//            return false; //can not be link-local, site-local or multicast address
      }
      else if ( (i + 1) == addrParts.length) {
         if ( num == 0 || num == 1)
            return false;   //can not be unspecified or loopback address
      }
      if ( num != 0 )
         break;
   }
   return true;
}

function isValidPrefixLength(prefixLen) {
   var num;

   num = parseInt(prefixLen);
   if (num <= 0 || num > 128)
      return false;
   return true;
}
```

```
function areSamePrefix(addr1, addr2) {
   var i, j;
   var a = [0, 0, 0, 0, 0, 0, 0, 0];
   var b = [0, 0, 0, 0, 0, 0, 0, 0];

   addr1Parts = addr1.split(':');
   if (addr1Parts.length < 3 || addr1Parts.length > 8)
      return false;
   addr2Parts = addr2.split(':');
   if (addr2Parts.length < 3 || addr2Parts.length > 8)
      return false;
   j = 0;
   for (i = 0; i < addr1Parts.length; i++) {
      if ( addr1Parts[i] == "" ) {
         if ((i != 0) && (i+1 != addr1Parts.length)) {
            j = j + (8 - addr1Parts.length + 1);
         }
         else {
            j++;
         }
      }
      else {
         a[j] = parseInt(addr1Parts[i], 16);
         j++;
      }
   }
   j = 0;
   for (i = 0; i < addr2Parts.length; i++) {
      if ( addr2Parts[i] == "" ) {
         if ((i != 0) && (i+1 != addr2Parts.length)) {
            j = j + (8 - addr2Parts.length + 1);
         }
         else {
            j++;
         }
      }
      else {
         b[j] = parseInt(addr2Parts[i], 16);
         j++;
      }
   }
   //only compare 64 bit prefix
   for (i = 0; i < 4; i++) {
      if (a[i] != b[i]) {
         return false;
      }
   }
   return true;
}

function getLeftMostZeroBitPos(num) {
   var i = 0;
   var numArr = [128, 64, 32, 16, 8, 4, 2, 1];

   for ( i = 0; i < numArr.length; i++ )
      if ( (num & numArr[i]) == 0 )
```

```
         return i;

   return numArr.length;
}

function getRightMostOneBitPos(num) {
   var i = 0;
   var numArr = [1, 2, 4, 8, 16, 32, 64, 128];

   for ( i = 0; i < numArr.length; i++ )
      if ( ((num & numArr[i]) >> i) == 1 )
         return (numArr.length - i - 1);

   return -1;
}

function isValidSubnetMask(mask) {
   var i = 0, num = 0;
   var zeroBitPos = 0, oneBitPos = 0;
   var zeroBitExisted = false;

   if ( mask == '0.0.0.0' )
      return false;

   maskParts = mask.split('.');
   if ( maskParts.length != 4 ) return false;

   for (i = 0; i < 4; i++) {
      if ( isNaN(maskParts[i]) == true )
         return false;
      num = parseInt(maskParts[i]);
      if ( num < 0 || num > 255 )
         return false;
      if ( zeroBitExisted == true && num != 0 )
         return false;
      zeroBitPos = getLeftMostZeroBitPos(num);
      oneBitPos = getRightMostOneBitPos(num);
      if ( zeroBitPos < oneBitPos )
         return false;
      if ( zeroBitPos < 8 )
         zeroBitExisted = true;
   }

   return true;
}

function isValidPort(port) {
   var fromport = 0;
   var toport = 100;

   portrange = port.split(':');
   if ( portrange.length < 1 || portrange.length > 2 ) {
       return false;
   }
   if ( isNaN(portrange[0]) )
       return false;
   fromport = parseInt(portrange[0]);
```

```
   if ( portrange.length > 1 ) {
       if ( isNaN(portrange[1]) )
          return false;
       toport = parseInt(portrange[1]);
       if ( toport <= fromport )
           return false;
   }

   if ( fromport < 1 || fromport > 65535 || toport < 1 || toport > 65535 )
       return false;

   return true;
}

function isValidNatPort(port) {
   var fromport = 0;
   var toport = 100;

   portrange = port.split('-');
   if ( portrange.length < 1 || portrange.length > 2 ) {
       return false;
   }
   if ( isNaN(portrange[0]) )
       return false;
   fromport = parseInt(portrange[0]);

   if ( portrange.length > 1 ) {
       if ( isNaN(portrange[1]) )
          return false;
       toport = parseInt(portrange[1]);
       if ( toport <= fromport )
           return false;
   }

   if ( fromport < 1 || fromport > 65535 || toport < 1 || toport > 65535 )
       return false;

   return true;
}

function isValidMacAddress(address) {
   var c = '';
   var num = 0;
   var i = 0, j = 0;
   var zeros = 0;

   addrParts = address.split(':');
   if ( addrParts.length != 6 ) return false;

   for (i = 0; i < 6; i++) {
      if ( addrParts[i] == '' )
         return false;
      for ( j = 0; j < addrParts[i].length; j++ ) {
         c = addrParts[i].toLowerCase().charAt(j);
         if ( (c >= '0' && c <= '9') ||
              (c >= 'a' && c <= 'f') )
```

```
            continue;
         else
            return false;
      }

      num = parseInt(addrParts[i], 16);
      if ( num == NaN || num < 0 || num > 255 )
         return false;
      if ( num == 0 )
         zeros++;
   }
   if (zeros == 6)
      return false;

   return true;
}

function isValidMacMask(mask) {
   var c = '';
   var num = 0;
   var i = 0, j = 0;
   var zeros = 0;
   var zeroBitPos = 0, oneBitPos = 0;
   var zeroBitExisted = false;

   maskParts = mask.split(':');
   if ( maskParts.length != 6 ) return false;

   for (i = 0; i < 6; i++) {
      if ( maskParts[i] == '' )
         return false;
      for ( j = 0; j < maskParts[i].length; j++ ) {
         c = maskParts[i].toLowerCase().charAt(j);
         if ( (c >= '0' && c <= '9') ||
              (c >= 'a' && c <= 'f') )
            continue;
         else
            return false;
      }

      num = parseInt(maskParts[i], 16);
      if ( num == NaN || num < 0 || num > 255 )
         return false;
      if ( zeroBitExisted == true && num != 0 )
         return false;
      if ( num == 0 )
         zeros++;
      zeroBitPos = getLeftMostZeroBitPos(num);
      oneBitPos = getRightMostOneBitPos(num);
      if ( zeroBitPos < oneBitPos )
         return false;
      if ( zeroBitPos < 8 )
         zeroBitExisted = true;
   }
   if (zeros == 6)
      return false;
```

```
   return true;
}

var hexVals = new Array("0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
              "A", "B", "C", "D", "E", "F");
var unsafeString = "\"<>%\\^[]`\+\$\,'#&";
// deleted these chars from the include list ";", "/", "?", ":", "@", "=", "&" and #
// so that we could analyze actual URLs

function isUnsafe(compareChar)
// this function checks to see if a char is URL unsafe.
// Returns bool result. True = unsafe, False = safe
{
   if ( unsafeString.indexOf(compareChar) == -1 && compareChar.charCodeAt(0) > 32
        && compareChar.charCodeAt(0) < 123 )
      return false; // found no unsafe chars, return false
   else
      return true;
}

function decToHex(num, radix)
// part of the hex-ifying functionality
{
   var hexString = "";
   while ( num >= radix ) {
      temp = num % radix;
      num = Math.floor(num / radix);
      hexString += hexVals[temp];
   }
   hexString += hexVals[num];
   return reversal(hexString);
}

function reversal(s)
// part of the hex-ifying functionality
{
   var len = s.length;
   var trans = "";
   for (i = 0; i < len; i++)
      trans = trans + s.substring(len-i-1, len-i);
   s = trans;
   return s;
}

function convert(val)
// this converts a given char to url hex form
{
   return  "%" + decToHex(val.charCodeAt(0), 16);
}



function encodeUrl(val)
{
   var len     = val.length;
   var i       = 0;
   var newStr  = "";
   var original = val;
```

```
   for ( i = 0; i < len; i++ ) {
      if ( val.substring(i,i+1).charCodeAt(0) < 255 ) {
         // hack to eliminate the rest of unicode from this
         if (isUnsafe(val.substring(i,i+1)) == false)
            newStr = newStr + val.substring(i,i+1);
         else
            newStr = newStr + convert(val.substring(i,i+1));
      } else {
         // woopsie! restore.
         alert ("Found a non-ISO-8859-1 character at position: " + (i+1) + ",\nPlease
         eliminate before continuing.");
         newStr = original;
         // short-circuit the loop and exit
         i = len;
      }
   }

   return newStr;
}

var markStrChars = "\"'";

// Checks to see if a char is used to mark begining and ending of string.
// Returns bool result. True = special, False = not special
function isMarkStrChar(compareChar)
{
   if ( markStrChars.indexOf(compareChar) == -1 )
      return false; // found no marked string chars, return false
   else
      return true;
}

// use backslash in front one of the escape codes to process
// marked string characters.
// Returns new process string
function processMarkStrChars(str) {
   var i = 0;
   var retStr = '';

   for ( i = 0; i < str.length; i++ ) {
      if ( isMarkStrChar(str.charAt(i)) == true )
         retStr += '\\';
      retStr += str.charAt(i);
   }

   return retStr;
}

// Web page manipulation functions

function showhide(element, sh)
{
    var status;
    if (sh == 1) {
        status = "block";
    }
```

```
    else {
        status = "none"
    }

    if (document.getElementById)
    {
        // standard
        document.getElementById(element).style.display = status;
    }
    else if (document.all)
    {
        // old IE
        document.all[element].style.display = status;
    }
    else if (document.layers)
    {
        // Netscape 4
        document.layers[element].display = status;
    }
}

// Load / submit functions

function getSelect(item)
{
    var idx;
    if (item.options.length > 0) {
        idx = item.selectedIndex;
        return item.options[idx].value;
    }
    else {
        return '';
    }
}

function setSelect(item, value)
{
    for (i=0; i<item.options.length; i++) {
        if (item.options[i].value == value) {
            item.selectedIndex = i;
            break;
        }
    }
}

function setCheck(item, value)
{
    if ( value == '1' ) {
         item.checked = true;
    } else {
         item.checked = false;
    }
}

function setDisable(item, value)
{
    if ( value == 1 || value == '1' ) {
```

```
        item.disabled = true;
    } else {
        item.disabled = false;
    }
}

function submitText(item)
{
    return '&' + item.name + '=' + item.value;
}

function submitSelect(item)
{
    return '&' + item.name + '=' + getSelect(item);
}


function submitCheck(item)
{
    var val;
    if (item.checked == true) {
        val = 1;
    }
    else {
        val = 0;
    }
    return '&' + item.name + '=' + val;
}
```