Free information   Useful Libraries in C#

Summary of useful library classes

Simple set handling and other classes that are useful in the development of applications using Coco/R.

```
class IntSet  { // simple set handling routines
  public IntSet()
  public IntSet(int ... members)
  public object Clone()
  public bool Equals(IntSet s)
  public void Fill()
  public void Clear()
  public void Incl(int i)
  public void Excl(int i)
  public bool Contains(int i)
  public bool Contains(IntSet i)
  public bool IsEmpty()
  public bool IsFull()
  public int Members()
  public IntSet Union(IntSet s)
  public IntSet Difference(IntSet s)
  public IntSet Intersection(IntSet s)
  public IntSet SymDiff(IntSet s)
  public void Write()
  public string ToString()
  public string ToCharSetString()
} // IntSet

public class OutFile {  // text file output
  public static OutFile StdOut
  public static OutFile StdErr
  public OutFile()
  public OutFile(string fileName)
  public bool OpenError()
  public void Write(string s)
  public void Write(object o)
  public void Write(byte o)
  public void Write(short o)
  public void Write(sbyte o)
  public void Write(long o)
  public void Write(bool o)
  public void Write(float o)
  public void Write(double o)
  public void Write(char o)
  public void WriteLine()
  public void WriteLine(string s)
  public void WriteLine(object o)
  public void WriteLine(byte o)
  public void WriteLine(sbyte o)
  public void WriteLine(short o)
  public void WriteLine(int o)
  public void WriteLine(long o)
  public void WriteLine(bool o)
  public void WriteLine(float o)
  public void WriteLine(double o)
  public void WriteLine(char o)
  public void Write(string o,  int width)
  public void Write(object o,  int width)
  public void Write(byte o,    int width)
  public void Write(sbyte o,   int width)
  public void Write(short o,   int width)
  public void Write(int o,     int width)
  public void Write(long o,    int width)
  public void Write(bool o,    int width)
  public void Write(float o,   int width)
  public void Write(double o,  int width)
  public void Write(char o,    int width)
  public void WriteLine(string o,  int width)
  public void WriteLine(object o,  int width)
  public void WriteLine(byte o,    int width)
  public void WriteLine(sbyte o,   int width)
  public void WriteLine(short o,   int width)
  public void WriteLine(int o,     int width)
  public void WriteLine(long o,    int width)
  public void WriteLine(bool o,    int width)
  public void WriteLine(float o,   int width)
  public void WriteLine(double o,  int width)
  public void WriteLine(char o,    int width)
  public void Close()
} // OutFile
```

```
public class InFile {    // text file input
  public static InFile StdIn
  public InFile()
  public InFile(string fileName)
  public bool OpenError()
  public int ErrorCount()
  public static bool Done()
  public void ShowErrors()
  public void HideErrors()
  public bool EOF()
  public bool EOL()
  public bool Error()
  public bool NoMoreData()
  public char ReadChar()
  public void ReadAgain()
  public void SkipSpaces()
  public void ReadLn()
  public string ReadString()
  public string ReadString(int max)
  public string ReadLine()
  public String ReadWord()
  public int ReadInt()
  public int ReadInt(int radix)
  public long ReadLong()
  public sbyte ReadSByte()
  public byte ReadByte()
  public int ReadShort()
  public float ReadFloat()
  public double ReadDouble()
  public bool ReadBool()
  public void Close()
} // InFile
```

Strings and Characters in C#

The following rather meaningless code illustrates various of the string and character manipulation methods that are available in C# and which will be found to be useful in developing translators.

```
using System.Text;   // for StringBuilder
using System;        // for Char

    char c, c1, c2;
    bool b, b1, b2;
    string s, s1, s2;
    int i, i1, i2;

    b = Char.IsLetter(c);                  // true if letter
    b = Char.IsDigit(c);                   // true if digit
    b = Char.IsLetterOrDigit(c);           // true if letter or digit
    b = Char.IsWhiteSpace(c);              // true if white space
    b = Char.IsLower(c);                   // true if lowercase
    b = Char.IsUpper(c);                   // true if uppercase
    c = Char.ToLower(c);                   // equivalent lowercase
    c = Char.ToUpper(c);                   // equivalent uppercase
    s = c.ToString();                      // convert to string
    i = s.Length;                          // length of string
    b = s.Equals(s1);                      // true if s == s1
    b = String.Equals(s1, s2);             // true if s1 == s2
    i = String.Compare(s1, s2);            // i = -1, 0, 1 if s1 < = > s2
    i = String.Compare(s1, s2, true);      // i = -1, 0, 1 if s1 < = > s2, ignoring case
    s = s.Trim();                          // remove leading/trailing whitespace
    s = s.ToUpper();                       // equivalent uppercase string
    s = s.ToLower();                       // equivalent lowercase string
    char[] ca = s.ToCharArray();           // create character array
    s = String.Concat(s1, s2);             // s1 + s2
    s = s.Substring(i1);                   // substring starting at s[i1]
    s = s.Substring(i1, i2);               // substring s[i1 ... i1+i2-1] (i2 is length)
    s = s.Remove(i1, i2);                  // remove i2 chars from s[i1]
    s = s.Replace(c1, c2);                 // replace all c1 by c2
    s = s.Replace(s1, s2);                 // replace all s1 by s2
    c = s[i];                              // extract i-th character of s
//    s[i] = c;                            // not allowed
    i = s.IndexOf(c);                      // position of c in s[0 ...
    i = s.IndexOf(c, i1);                  // position of c in s[i1 ...
    i = s.IndexOf(s1);                     // position of s1 in s[0 ...
    i = s.IndexOf(s1, i1);                 // position of s1 in s[i1 ...
    i = s.LastIndexOf(c);                  // last position of c in s
    i = s.LastIndexOf(c, i1);              // last position of c in s, <= i1
    i = s.LastIndexOf(s1);                 // last position of s1 in s
    i = s.LastIndexOf(s1, i1);             // last position of s1 in s, <= i1
    i = Convert.ToInt32(s);                  // convert string to integer
    i = Convert.ToInt32(s, i1);              // convert string to integer, base i1
    s = Convert.ToString(i);                 // convert integer to string

    StringBuilder                            // build strings
      sb = new StringBuilder(),              //
      sb1 = new StringBuilder("original");   //
    sb.Append(c);                            // append c to end of sb
    sb.Append(s);                            // append s to end of sb
    sb.Insert(i, c);                         // insert c in position i
    sb.Insert(i, s);                         // insert s in position i
    b = sb.Equals(sb1);                      // true if sb == sb1
    i = sb.Length;                           // length of sb
    sb.Remove(i1, i2);                       // remove i2 chars from sb[i1]
    sb.Replace(c1, c2);                      // replace all c1 by c2
    sb.Replace(s1, s2);                      // replace all s1 by s2
    s = sb.ToString();                       // convert sb to real string
    c = sb[i];                               // extract sb[i]
    sb[i] = c;                               // sb[i] = c

    char[] delim = new char[] {'a', 'b'};
    string[]  tokens;                        // tokenize strings
    tokens = s.Split(delim);                 // delimiters are a and b
    tokens = s.Split('.' ,':', '@');         // delimiters are . : and @
    tokens = s.Split(new char[] {'+', '-'}); // delimiters are + -?
    for (int i = 0; i < tokens.Length; i++)  // process successive tokens
      Process(tokens[i]);
  }
}
```

Simple graphics in C#

The following is the specification of a very simple C# (2.0/3.0) graphics drawing library.

```
using GraphicsLib;

public class Turtle {
// Basic TurtleGraphics Library for use with Parva and the PVM (C#)

  public static void InitGraphics(int width, int height)
  // Initialize drawing canvas width * height pixels
  // and locate turtle at mid point of this canvas, facing East

  public static void CloseGraphics()
  // Dispose of drawing canvas

  public static void Home()
  // Move the turtle back to its original position, facing East

  public static void PenUp()
  // Raise the turtle's pen, so as to inhibit drawing on the canvas

  public static void PenDown()
  // Raise the turtle's pen, so as to produce drawing on the canvas

  public static void TurnLeft(int degrees)
  // Turn the turtle left through the specified number of degrees (without moving)

  public static void Forward(int distance)
  // Move the turtle the specified pixel distance in the direction it is pointing

  public static void Line(int x1, int y1, int x2, int y2)
  // Draw a line segment from (x1, y1) to (x2, y2)

} // class Turtle
```

Simple list handling in C#

The following is the specification of useful members of a C# (2.0/3.0) list handling class.

```
using System.Collections.Generic;

class List
// Class for constructing a list of elements of type E

  public List<E> ()
  // Empty list constructor

  public int Add(E element)
  // Appends element to end of list
  public element this [int index] {set; get; }
  // Inserts or retrieves an element in position index
  // list[index] = element;  element = list[index]

  public void Clear()
  // Clears all elements from list

  public int Count { get; }
  // Returns number of elements in list

  public bool Contains(E element)
  // Returns true if element is in the list

  public int IndexOf(E element)
  // Returns position of element in the list

  public void Remove(E element)
  // Removes element from list

  public void RemoveAt(int index)
  // Removes the element at position index

} // List
```

Simple dictionary handling in C#

The following is the specification of useful members of a C# (2.0/3.0) dictionary handling class.

```
using System.Collections.Generic;

class Dictionary<K, V>
// Class for constructing a dictionary (map) of elements with keys of type K and values of type V

  public Dictionary<K, V>()
  // Empty dictionary constructor

  public Dictionary<K, V>(int capacity)
  // Empty dictionary with specified initial capacity

  public void Clear()
  // Clears all elements from dictionary

  public int Count( get; }
  // Returns number of elements in dictionary

  public void Add(K key, V value)
  // Adds (K, V) to dictionary (replacing any existing entry with that key)

  V this [K key]  { set; }
  // Indexer - adds new element, or replaces existing element, using specified key

  V this [K key]  { get; }
  // Indexer - retrieves element using specified key, or raises exception if not present
  // (Protect yourself by using ContainsKey or TryGetValue!)

  // public bool IsEmpty() - not a member, unfortunately

  public bool ContainsKey(K key)
  // Returns true if element with specified key is in the dictionary

  public bool ContainsValue(V value)
  // Returns true if element with specified value is in the dictionary

  public bool TryGetValue(K key, out V value)
  // Returns true if element with specified key is in the dictionary, returning that value
  // Returns false if element with specified key is  notin the dictionary, returning null

  public Remove(K key)
  // Removes the element with specified key from dictionary (no effect if no such element exists)

  public ICollection<K> Keys { get; }
  // Returns a set of all the keys in the dictionary (useful for iterations)

  public ICollection<V> Values { get; }
  // Returns a collection of all the values in the dictionary (useful for iterations)

  // Iterating drectly over a generic dictionary interface returns a sequence of KeyValuePair
  // constructs
  //
  // public struct KeyValuePair <K, V> {
  //    public K Key   { get ; }
  //    public V Value { get ; }
  // } // KeyValuePair

} // Dictionary
```

Simple set handling in C#

The following is the specification of useful members of a C# (2.0/3.0) integer set handling class.

```
class IntSet
// Simple set class
// P.D. Terry (p.terry@ru.ac.za)

  public IntSet()
  // Empty set constructor

  public IntSet(params int[] members)
  // Variable args constructor  IntSet(a)  IntSet(a, b) etc

  public IntSet Clone()
  // Value copy

  public bool Equals(IntSet that)
  // Value comparison of this set with that set

  public void Incl(int i)
  // Includes i in this set

  public void Excl(int i)
  // Excludes i from this set

  public bool Contains(int i)
  // Returns true if i is a member of this set

  public bool Contains(IntSet that) {
  // Returns true if that set is a subset of this set

  public bool IsEmpty()
  // Returns true if this set is empty

  public bool IsFull() {
  // Returns true if this set is a universe set

  public void Fill()
  // Creates a full universe set for this set

  public void Clear() {
  // Clear this set

  public int Members()
  // Returns number of members in this set

  public IntSet Union(IntSet that)
  // Returns union of this set with that set

  public IntSet Difference(IntSet that)
  // Returns difference of this set with that set

  public IntSet Intersection(IntSet that)
  // Returns intersection of this set with that set

  public IntSet SymDiff(IntSet that)
  // Returns symmetric difference of this set with that set

  public override string ToString()
  // Returns string representation of this set as a set of ints

  public string ToCharSetString() {
  // Returns string representation of this set as a set of chars

} // IntSet
```