Bzip2 checksum algorithm

```
// StrangeCRC.cs - computes a crc used in the bziplib
//
// Copyright (C) 2001 Mike Krueger
//
// This file was translated from java, it was part of the GNU Classpath
// Copyright (C) 1999, 2000, 2001 Free Software Foundation, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
//
// Linking this library statically or dynamically with other modules is
// making a combined work based on this library. Thus, the
```

```
terms and
// conditions of the GNU General Public License cover the whole
// combination.
//
// As a special exception, the copyright holders of this library give you
// permission to link this library with independent modules to produce an
// executable, regardless of the license terms of these independent
// modules, and to copy and distribute the resulting executable under
// terms of your choice, provided that you also meet, for each linked
// independent module, the terms and conditions of the license of that
// module. An independent module is a module which is not derived from
// or based on this library. If you modify this library, you may extend
// this exception to your version of the library, but you are not
// obligated to do so. If you do not wish to do so, delete this
// exception statement from your version.

using System;

namespace ICSharpCode.SharpZipLib.Checksums
{
///

/// Bzip2 checksum algorithm
///
public class StrangeCRC
{
readonly static uint[] crc32Table = {
```

```
0x00000000, 0x04c11db7, 0x09823b6e, 0x0d4326d9,
0x130476dc, 0x17c56b6b, 0x1a864db2, 0x1e475005,
0x2608edb8, 0x22c9f00f, 0x2f8ad6d6, 0x2b4bcb61,
0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbdbd,
0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9,
0x5f15adac, 0x5bd4b01b, 0x569796c2, 0x52568b75,
0x6a1936c8, 0x6ed82b7f, 0x639b0da6, 0x675a1011,
0x791d4014, 0x7ddc5da3, 0x709f7b7a, 0x745e66cd,
0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5,
0xbe2b5b58, 0xbaea46ef, 0xb7a96036, 0xb3687d81,
0xad2f2d84, 0xa9ee3033, 0xa4ad16ea, 0xa06c0b5d,
0xd4326d90, 0xd0f37027, 0xddb056fe, 0xd9714b49,
0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1,
0xe13ef6f4, 0xe5ffeb43, 0xe8bccd9a, 0xec7dd02d,
0x34867077, 0x30476dc0, 0x3d044b19, 0x39c556ae,
0x278206ab, 0x23431b1c, 0x2e003dc5, 0x2ac12072,
0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcdbb16,
0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca,
0x7897ab07, 0x7c56b6b0, 0x71159069, 0x75d48dde,
0x6b93dddb, 0x6f52c06c, 0x6211e6b5, 0x66d0fb02,
0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1, 0x53dc6066,
0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e,
0xbfa1b04b, 0xbb60adfc, 0xb6238b25, 0xb2e29692,
0x8aad2b2f, 0x8e6c3698, 0x832f1041, 0x87ee0df6,
0x99a95df3, 0x9d684044, 0x902b669d, 0x94ea7b2a,
0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2,
0xc6bcf05f, 0xc27dede8, 0xcf3ecb31, 0xcbffd686,
0xd5b88683, 0xd1799b34, 0xdc3abded, 0xd8fba05a,
0x690ce0ee, 0x6dcdfd59, 0x608edb80, 0x644fc637,
0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f,
0x5c007b8a, 0x58c1663d, 0x558240e4, 0x51435d53,
0x251d3b9e, 0x21dc2629, 0x2c9f00f0, 0x285e1d47,
```

```
0x36194d42, 0x32d850f5, 0x3f9b762c, 0x3b5a6b9b,
0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623,
0xf12f560e, 0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7,
0xe22b20d2, 0xe6ea3d65, 0xeba91bbc, 0xef68060b,
0xd727bbb6, 0xd3e6a601, 0xdea580d8, 0xda649d6f,
0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7,
0xae3afba2, 0xaafbe615, 0xa7b8c0cc, 0xa379dd7b,
0x9b3660c6, 0x9ff77d71, 0x92b45ba8, 0x9675461f,
0x8832161a, 0x8cf30bad, 0x81b02d74, 0x857130c3,
0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
0x4e8ee645, 0x4a4ffbf2, 0x470cdd2b, 0x43cdc09c,
0x7b827d21, 0x7f436096, 0x7200464f, 0x76c15bf8,
0x68860bfd, 0x6c47164a, 0x61043093, 0x65c52d24,
0x119b4be9, 0x155a565e, 0x18197087, 0x1cd86d30,
0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088,
0x2497d08d, 0x2056cd3a, 0x2d15ebe3, 0x29d4f654,
0xc5a92679, 0xc1683bce, 0xcc2b1d17, 0xc8ea00a0,
0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb, 0xdbee767c,
0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xeee2ed18,
0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4,
0x89b8fd09, 0x8d79e0be, 0x803ac667, 0x84fbdbd0,
0x9abc8bd5, 0x9e7d9662, 0x933eb0bb, 0x97ffad0c,
0xafb010b1, 0xab710d06, 0xa6322bdf, 0xa2f33668,
0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
};

int globalCrc;

///

/// Initialise a default instance of
///
public StrangeCRC()
{
Reset();
```

```
}

///

/// Reset the state of Crc.
///
public void Reset()
{
globalCrc = -1;
}

///

/// Get the current Crc value.
///
public long Value {
get {
return ~globalCrc;
}
}

///

/// Update the Crc value.
///
/// data update is based on public void Update(int value)
{
int temp = (globalCrc >> 24) ^ value;
if (temp < 0) { temp = 256 + temp; } globalCrc =
unchecked((int)((globalCrc < /// Update Crc based on a block
of data
///

/// The buffer containing data to update the crc with. public
void Update(byte[] buffer)
{
if (buffer == null) {
throw new ArgumentNullException(“buffer”);
}
```

```
Update(buffer, 0, buffer.Length);
}

///

/// Update Crc based on a portion of a block of data
///
/// block of data /// index of first byte to use /// number of bytes to use public void Update(byte[] buffer, int offset, int count)
{
if (buffer == null) {
throw new ArgumentNullException(“buffer”);
}

if ( offset < 0 ) { #if NETCF_1_0 throw new ArgumentOutOfRangeException("offset"); #else throw new ArgumentOutOfRangeException("offset", "cannot be less than zero"); #endif } if ( count < 0 ) { #if NETCF_1_0 throw new ArgumentOutOfRangeException("count"); #else throw new ArgumentOutOfRangeException("count", "cannot be less than zero"); #endif } if ( offset + count > buffer.Length )
{
throw new ArgumentOutOfRangeException(“count”);
}

for (int i = 0; i < count; ++i) { Update(buffer[offset++]); }
} } } [/csharp]
```