Subject: new: CRC handling class
Posted by kohait00 on Wed, 28 Mar 2012 21:43:50 GMT
View Forum Message <> Reply to Message

hi folks

due to some work with CRC recently, i realized that most implementations are really opaque and info is rare. implementations, where the algorithm is visible are rare too.
so i came up with a template crc class. currently its crc16 only
but it will soon be extended with crc32. the different polynoms can be searched in wiki and added as needed, since the algorithm doesnt change as for byte by byte processing. it uses a 256 values lookup table.

feel free to contribute ideas. it's surely not the most advanced and most optimized code, but the benefits of unreadable code are neglectable with todays computing power, right

maybe this is of use for somebody else..i could add in bazaar, or it can go in Core as well, could be of some use for Serialize..

the class looks like this (lsbf is also available)

```
template<word P>
class CRC16MSBF
{
public:

 class Table
 {
 public:
  static const word size = 1 << 8;
  Table()
  {
   for(word i = 0; i < size; i++)
   {
    word crc = i << 8;
    for(int b = 0; b < 8; b++)
     crc = (crc & 0x8000) ? ((crc << 1) ^ P) : (crc << 1);
    t[i] = crc;
   }
  }
  word t[size];
 };

 CRC16MSBF (word init = word(-1))
  : t(Single<Table>())
 { Init(init); }

 inline void Add(byte data) { crc = (crc << 8) ^ t.t[ (crc >> 8) ^ data ]; }
```

```
 void Add(const byte* pdata, int size)
 {
  const byte* pde = pdata + size;
  while(pdata < pde) Add(*pdata++);
 }

 inline CRC16MSBF& operator <<= (byte data) { Add(data); return *this; }

#ifdef CPU_LE
 word Value() const { return word( (crc >> 8) | (crc << 8) ); }
#endif
#ifdef CPU_BE
 word Value() const { return crc; }
#endif

 word Raw() const { return crc; }
 String Serialized() const { String s; word c = Value(); s.Cat(c & 0xFF); s.Cat(c >> 8); return s; }

 bool IsValid() const { return (crc == 0); }
 word Invalidate() { return ++crc; }

 void Init(word in) { crc = init = in; }
 void Reset() { crc = init; }

 const Table& t;
 word crc;
 word init;
};
```

and the test case

```
void TestCRC()
{
 String s = "This is a soon to be protected string!";

 CRC16LSBF_CCITT crc; //a common init value must be used, generally 0xFFFF..

 //process some data, more data could be added
 crc.Add((const byte*)s.Begin(), s.GetLength());

 //append crc value to the protected string if it should be included in check later
 //this is typical for hardware implementations.
 //software usually tests for equality, but it is esier in first case
 //since a packet can be checked as a whole.
 s << crc.Serialized();
```

```
 //transmit s, receiver checks it

 crc.Reset(); //here we just reuse the crc with its specified init value

 //check now includes the crc value appended
 crc.Add((const byte*)s.Begin(), s.GetLength());

 ASSERT(crc.IsValid());

 //imagine a corrupt data stream
 s.Set(0,0);

 crc.Reset(); //here we just reuse the crc

 //check now includes the crc value appended
 crc.Add((const byte*)s.Begin(), s.GetLength());

 //now the stream must be invalid
 ASSERT(!crc.IsValid());
}
```

attached is the test environment

cheers

File Attachments

1) CRC.rar, downloaded 240 times