The Adler-32 algorithm

The Adler-32 algorithm is much faster than the CRC32 algorithm yet still provides an extremely low probability of undetected errors.

The modulo on unsigned long accumulators can be delayed for 5552 bytes, so the modulo operation time is negligible. If the bytes are a, b, c, the second sum is 3a + 2b + c + 3, and so is position and order sensitive, unlike the first sum, which is just a checksum. That 65521 is prime is important to avoid a possible large class of two-byte errors that leave the check unchanged. (The Fletcher checksum uses 255, which is not prime and which also makes the Fletcher check insensitive to single byte changes 0 <-> 255.)

The sum s1 is initialized to 1 instead of zero to make the length of the sequence part of s2, so that the length does not have to be checked separately. (Any sequence of zeroes has a Fletcher checksum of zero.)

Appendix: Sample code

The following C code computes the Adler-32 checksum of a data buffer. It is written for clarity, not for speed. The sample code is in the ANSI C programming language. Non C users may find it easier to read with these hints:

```
&      Bitwise AND operator.
>>     Bitwise right shift operator. When applied to an
       unsigned quantity, as here, right shift inserts zero bit(s)
       at the left.
<<     Bitwise left shift operator. Left shift inserts zero
       bit(s) at the right.
++     "n++" increments the variable n.
%      modulo operator: a % b is the remainder of a divided by b.
```

```
#define BASE 65521 /* largest prime smaller than 65536 */

/*
   Update a running Adler-32 checksum with the bytes buf[0..len-1]
 and return the updated checksum. The Adler-32 checksum should be
 initialized to 1.

 Usage example:

   unsigned long adler = 1L;

   while (read_buffer(buffer, length) != EOF) {
     adler = update_adler32(adler, buffer, length);
   }
   if (adler != original_adler) error();
*/
unsigned long update_adler32(unsigned long adler,
   unsigned char *buf, int len)
{
  unsigned long s1 = adler & 0xffff;
  unsigned long s2 = (adler >> 16) & 0xffff;
  int n;

  for (n = 0; n < len; n++) {
    s1 = (s1 + buf[n]) % BASE;
    s2 = (s2 + s1)     % BASE;
  }
  return (s2 << 16) + s1;
}

/* Return the adler32 of the bytes buf[0..len-1] */

unsigned long adler32(unsigned char *buf, int len)
{
  return update_adler32(1L, buf, len);
}
```