EAStdC 1.03.06 Index

namespace EA::StdC

v1.0

Luc Isaak

EAStdC written and maintained by:
Paul Pedriana

EAAlignment.h

```
#define   EAAlignOf
size_t    AlignOf<T>
size_t    AlignOf<T>

T         AlignUp<T, size_t a>
T*        AlignUp<T, size_t a>
T         AlignUp<T>
T*        AlignUp<T>
T         AlignDown<T, size_t a>
T*        AlignDown<T, size_t a>
T         AlignDown<T>
T*        AlignDown<T>
size_t    GetAlignment<T>
bool      IsAligned<T, size_t a>
bool      IsAligned<T>

template<T, int count,
          int alignment>
class     AlignedArray

template<T, int alignment>
class     AlignedObject

uint16_t ReadMisalignedUint16
uint32_t ReadMisalignedUint32
uint64_t ReadMisalignedUint64
void      WriteMisalignedUint16
void      WriteMisalignedUint32
void      WriteMisalignedUint64
```

EABitTricks.h

```
// Bit manipulation
T         TurnOffLowestBit<T>
T         IsolateLowestBit<T>
T         IsolateLowest0Bit<T>
T         GetTrailing0Bits<T>
T         GetTrailing1And0Bits<T>
T
   PropogateLowestBitDownward<T>
T
   TurnOffLowestContiguousBits<T>
T         TurnOnLowest0Bit<T>
unsigned GetNextWithEqualBitCount
unsigned IsolateSingleBits
unsigned IsolateSingle0Bits
unsigned IsolateSingle0And1Bits
int32_t   ShiftRightSigned
int       CountTrailing0Bits
int       CountLeading0Bits
int       CountBits
int       CountBits64

uint32_t RotateLeft
uint32_t RotateRight
uint32_t ReverseBits
uint32_t IsolateHighestBit
uint32_t IsolateHighest0Bit
uint32_t PropogateHighestBitDownward
uint32_t GetHighestContiguous0Bits
T         GetBitwiseEquivalence<T>
bool      AreLessThan2BitsSet<T>
T         GetHighestBit<T>
bool      IsPowerOf2<T>
uint32_t RoundUpToPowerOf2<T>
bool      IsPowerOf2Multiple<T,int n>
bool      IsPowerOf2Minus1<T>
bool      CrossesPowerOf2<T>
bool      CrossesPowerOf2<T, int n>
uint32_t GetHighestBitPowerOf2
uint32_t GetHighestBitPowerOf2
T         GetNextGreaterEven<T>
T         GetNextGreaterOdd<T>
T         RoundUpTo<T, int n>
int32_t   RoundUpToEx<T, int n>
T         RoundDownTo<T, int n>
T         RoundDownToEx<T, int n>
T         RoundUpToMultiple<T, int n>
T         RoundDownToMultiple<T, int>
uint32_t Log2
uint32_t CeilLog2

// Overflow
bool
   SignedAdditionWouldOverflow<T>
bool
   SignedSubtractionWouldOverflow<T>
bool
    UnsignedAdditionWouldOverflow<T>
bool
 UnsignedSubtractionWouldOverflow<T>
bool  UnsignedMultiplyWouldOverflow
bool  SignedMultiplyWouldOverflow
bool  UnsignedDivisionWouldOverflow
bool  SignedDivisionWouldOverflow
int   GetAverage
int   GetAverage_Ceiling

// Miscellaneous
int       GetParity
bool      GetIsBigEndian
int       ToggleBetween0And1
int       ToggleBetweenIntegers<T>
T
bool      IsBetween0AndValue
void      ExchangeValues<T>
T         FloorMod<T>
int       GetSign
int       GetSignEx
int32_t   SignExtend12
int32_t   SignExtend24
bool      IsUnsigned<T>
#define   EAIsUnsigned
bool      IsTwosComplement
bool      IsOnesComplement
bool      IsSignMagnitude
bool      IsOffsetBinary
#define   EAArrayCount
#define   EAOffsetOf
```

EAByteCrackers.h

Conventions:
0 (zero) refers to the lowest byte,
1 refers to the second lowest byte, etc.
b means   8 bit byte
w means 16 bit word
d means 32 bit dword
q means 64 bit quadword

```
#define UINT8_0_FROM_UINT16(w)
#define UINT8_1_FROM_UINT16(w)

#define UINT8_0_FROM_UINT32(d)
#define UINT8_1_FROM_UINT32(d)
#define UINT8_2_FROM_UINT32(d)
#define UINT8_3_FROM_UINT32(d)

#define UINT8_0_FROM_UINT64(q)
#define UINT8_1_FROM_UINT64(q)
#define UINT8_2_FROM_UINT64(q)
#define UINT8_3_FROM_UINT64(q)
#define UINT8_4_FROM_UINT64(q)
#define UINT8_5_FROM_UINT64(q)
#define UINT8_6_FROM_UINT64(q)
#define UINT8_7_FROM_UINT64(q)

#define UINT16_0_FROM_UINT32(d)
#define UINT16_1_FROM_UINT32(d)

#define UINT16_0_FROM_UINT64(q)
#define UINT16_1_FROM_UINT64(q)
#define UINT16_2_FROM_UINT64(q)
#define UINT16_3_FROM_UINT64(q)

#define UINT16_FROM_UINT8(b1, b0)

#define UINT32_0_FROM_UINT64(q)
#define UINT32_1_FROM_UINT64(q)

#define UINT32_FROM_UINT8(b3, b2,
                          b1, b0)
#define UINT32_FROM_UINT16(w1, w0)

#define UINT64_FROM_UINT8( b7, b6,
                           b5, b4,
                           b3, b2,
                           b1, b0)
#define UINT64_FROM_UINT16(w3, w2,
                           w1, w0)
#define UINT64_FROM_UINT32(d1, d0)
```

eactype.h

```
int       Isalnum
int       Isalpha
int       Isdigit
int       Isxdigit
int       Isgraph
int       Islower
int       Isupper
int       Isprint
int       Ispunct
int       Isspace
int       Iscntrl
int       Isascii

char_t    Tolower
char_t    Toupper(char_t c)
```

EADateTime.h

```
class DateTime
{
 DateTime(TimeFrame)
 DateTime(int64_t nSeconds)
 DateTime(uint32_t nYear,
          uint32_t nMonth,
          uint32_t nDayOfMonth,
          uint32_t nHour   = 0,
          uint32_t nMinute = 0,
          uint32_t nSecond = 0)

 int      Compare(const DateTime&,
                  bool  bCompDate,
                  bool  bCompTime)

 uint32_t GetParameter
 void     SetParameter
 void     Set(TimeFrame)
 void     Set(uint32_t  nYear,
              uint32_t  nMonth,
              uint32_t  nDayOfMonth,
              uint32_t  nHour,
              uint32_t  nMinute,
              uint32_t  nSecond)
 void      AddTime
 int64_t   GetSeconds
}

bool      IsLeapYear
uint32_t  GetDaysInYear
uint32_t  GetDaysInMonth
uint32_t  GetDayOfYear
int       Convert4DigitTo2DigitYear
int       Convert2DigitTo4DigitYear
uint32_t  GetCurrent
bool      IsDST
int64_t   GetDaylightSavingsBias
int64_t   GetTimeZoneBias

void      DateTimeToTm
void      TmToDateTime
void      DateTimeToFileTime
void      FileTimeToDateTime
void      DateTimeToSystemTime
void      SystemTimeToDateTime

bool      operator==
bool      operator>
bool      operator>=
bool      operator<
bool      operator<=
bool      operator!=
```

EAFixedPoint.h

```
typedef int32_t        EAFixed16
// 16:16 fixed point (16 bits of fraction)
typedef FPTemplate<…> SFixed16
typedef FPTemplate<…> UFixed16

#define       EAFixed16ToInt
#define       EAIntToFixed16
#define       EAFixed16ToDouble
#define       EADoubleToFixed16
#define       EAFixed16ToFloat
#define       EAFloatToFixed16
#define       EAFixed16Negate

EAFixed16     EAFixed16Mul
EAFixed16     EAFixed16Div
EAFixed16     EAFixed16DivSafe
EAFixed16     EAFixed16MulDiv
EAFixed16     EAFixed16MulDivSafe
EAFixed16     EAFixed16Mod
EAFixed16     EAFixed16ModSafe
EAFixed16     EAFixed16Abs

template <T, int upShiftInt,
      int downShiftInt, int upMulInt,
      int downDivInt>
struct FPTemplate
{
 void           FromFixed
 T              AsFixed
 int            AsInt
 unsigned int   AsUnsignedInt
 long           AsLong
 unsigned long  AsUnsignedLong
 float          AsFloat
 double         AsDouble

 bool           operator<
 bool           operator>
 bool           operator>=
 bool           operator<=
 bool           operator==
 bool           operator!=
 FPTemplate     operator~
 FPTemplate     operator-
 FPTemplate     operator+
 FPTemplate&    operator+=
 FPTemplate&    operator-=
 FPTemplate&    operator*=
 FPTemplate&    operator/=
 FPTemplate&    operator%=
 FPTemplate&    operator|=
 FPTemplate&    operator&=
 FPTemplate&    operator^=
 FPTemplate&    operator^=
 FPTemplate     operator<<
 FPTemplate     operator>>
 FPTemplate&    operator<<=
 FPTemplate&    operator>>=
 FPTemplate&    operator++
 FPTemplate&    operator--
 FPTemplate     operator++
 FPTemplate     operator--

 FPTemplate     Abs
 FPTemplate     DivSafe
 FPTemplate&    DivSafeAssign
 static T       FixedMul
 static T       FixedDiv
 static T       FixedDivSafe
 static T       FixedMulDiv
 static T       FixedMulDivSafe
 static T       FixedMod
 static T       FixedModSafe
}
```

EAGlobal.h

```
template<T, uint32_t kGlobalId =
T::kGlobalId>
class GlobalPtr

struct OSGlobalNode : public
EA::StdC::intrusive_list_node

typedef OSGlobalNode
*(*OSGlobalFactoryPtr)()

OSGlobalNode* GetOSGlobal
bool          SetOSGlobal
bool          ReleaseOSGlobal

template<T, uint32_t id>
class AutoOSGlobalPtr

template<T, uint32_t id>
class AutoStaticOSGlobalPtr
```

EAHashCRC.h

```
uint16_t CRC16
uint32_t CRC24
uint32_t CRC32
uint32_t CRC32Reverse
uint64_t CRC64
```

EAHashString.h

```
uint32_t FNV1
uint32_t FNV1_String8
uint32_t FNV1_String16

uint32_t DJB2
uint32_t DJB2_String8
uint32_t DJB2_String16

// Compile-time string hash. Generates
FNV1 string hashes via compile-time
template expansion. Both single byte and
Unicode characters are accepted.

template <>
class CTStringHash
```

EAMathHelp.h

```
uint32_t RoundToUint32
int32_t  RoundToInt32
int32_t  FloorToInt32
int32_t  CeilToInt32
int32_t  TruncateToInt32
int32_t  FastRoundToInt23
uint8_t  UnitFloatToUint8
uint8_t  ClampUnitFloatToUint8

bool     IsInvalid
bool     IsInvalid
bool     IsNormal
bool     IsNormal
bool     IsNAN
bool     IsNAN
bool     IsInfinite
bool     IsInfinite
bool     IsIndefinite
bool     IsIndefinite
bool     IsDenormalized
bool     IsDenormalized
```

EAMemory.h

```
void*          alloca
uint8_t*       Memset8
uint16_t*      Memset16
uint32_t*      Memset32
uint64_t*      Memset64
void*          MemsetN
void*          MemsetPointer
const char_t*  Memchr
int            Memcmp
char_t*        Memcpy
char_t*        Memmove
```

EARandom.h

```
void          GetRandomSeed

class RandomLinearCongruential
class RandomMersenneTwister
{
 uint32_t     GetSeed
 void         SetSeed
 uint32_t     operator()
 uint32_t     RandomUint32Uniform
 double       RandomDoubleUniform
}

typedef RandomLinearCongruential
typedef RandomLinearCongruential
                          Random
                          RandomSmall
                          RandomFast
typedef RandomMersenneTwister
                          RandomQuality
```

EARandomDistribution.h

```
bool         RandomBool<Random>
int32_t      Random2<Random>
int32_t      Random4<Random>
int32_t      Random8<Random>
int32_t      Random16<Random>
int32_t      Random32<Random>
int32_t      Random64<Random>
int32_t      Random128<Random>
int32_t      Random256<Random>
int32_t      RandomPowerOfTwo<Random>
int32_t
 RandomInt32UniformRange<Random>
double
 RandomDoubleUniformRange<Random>
uint32_t
 RandomUint32WeightedChoice<Random>
int32_t
 RandomInt32GaussianRange<Random>
Float
 RandomFloatGaussianRange<Random,
                          Float>
int32_t
 RandomInt32TriangleRange<Random>
Float
 RandomFloatTriangleRange<Random,
                          Float>
```

EAScanf.h

```
int      Cscanf
int      Fscanf
int      Scanf
int      Sscanf
int      Vcscanf
int      Vfscanf
int      Vscanf
int      Vsscanf
```

EASprintf.h

```
int      Cprintf
int      Fprintf
int      Printf
int      Sprintf
int      Snprintf
int      Vcprintf
int      Vfprintf
int      Vprintf
int      Vsprintf
int      Vsnprintf
```

EAString.h

```
// Consider using Strlcpy as a safe
alternative to Strcpy or Strncpy
char_t*  Strcpy
char_t*  Strncpy
size_t   Strlcpy

// Consider using Strlcat as a safe
alternative to Strcat, StringnCat or Strncat
char_t*  Strcat
char_t*  Strncat
char_t*  StringnCat

size_t   Strlcat
size_t   Strlen
size_t   StrlenUTF8Decoded
size_t   StrlenUTF8Encoded

char_t*  Strend
char_t*  Strdup
void     Strdel
char_t*  Strupr
char_t*  Strlwr
char_t*  Strmix
char_t*  Strchr
size_t   Strcspn
size_t   Strcspn
char_t*  Strpbrk
char_t*  Strrchr
size_t   Strspn
char_t*  Strstr
char_t*  Stristr
char_t*  Strrstr
char_t*  Strirstr
char_t*  Strtok
const char_t*
         Strtok2
char_t*  Strset
char_t*  Strnset
char_t*  Strrev
int      Strcmp
int      Strncmp
int      Stricmp
int      Strnicmp
int      StrcmpAlnum
int      StricmpAlnum

char_t*  EcvtBuf
char_t*  FcvtBuf

char_t*  I32toa
char_t*  U32toa
char_t*  I64toa
char_t*  U64toa
double   Strtod
double   StrtodEnglish
int32_t  StrtoI32
uint32_t StrtoU32
int64_t  StrtoI64
uint64_t StrtoU64

int32_t  AtoI32
uint32_t AtoU32
int64_t  AtoI64
uint64_t AtoU64
double   Atof
double   AtofEnglish
char_t*  Ftoa
char_t*  FtoaEnglish
size_t   ReduceFloatString
```

EATextUtil.h

```
bool  WildcardMatch
bool  ParseDelimitedText
void  ConvertBinaryDataToASCIIArray
bool  ConvertASCIIArrayToBinaryData
const char_t*
      GetTextLine

bool  SplitTokenDelimited
bool  SplitTokenSeparated

int   BoyerMooreSearch
```

Int128_t.h

```
class int128_t_base
{
 void      operatorPlus
 void      operatorMinus
 void      operatorMul
 void      operatorShiftRight
 void      operatorShiftLeft

 bool      operator!
 void      operatorXOR
 void      operatorOR
 void      operatorAND

 bool      AsBool
 uint8_t   AsUint8
 uint16_t  AsUint16
 uint32_t  AsUint32
 uint64_t  AsUint64

 int       GetBit
 void      SetBit
 uint8_t   GetPartUint8
 uint16_t  GetPartUint16
 uint32_t  GetPartUint32
 uint64_t  GetPartUint64
 void      SetPartUint8
 void      SetPartUint16
 void      SetPartUint32
 void      SetPartUint64

 bool      IsZero
 void      SetZero
 void      TwosComplement
 void      InverseTwosComplement
}

class int128_t : int128_t_base
class uint128_t: int128_t_base
{
 int128_t()
 int128_t(uint32_t nPart0,
          uint32_t nPart1,
          uint32_t nPart2,
          uint32_t nPart3)
 int128_t(uint64_t nPart0,
          uint64_t nPart1)
 int128_t(int8_t   value)
 int128_t(uint8_t  value)
 int128_t(int16_t  value)
 int128_t(uint16_t value)
 int128_t(int32_t  value)
 int128_t(uint32_t value)
 int128_t(int64_t  value)
 int128_t(uint64_t value)
 int128_t(float    value)
 int128_t(double   value)
 int128_t(const char_t*,
         int nBase = 10)

 // Unary arithmetic/logic operators
 int128_t   operator-
 int128_t&  operator++
 int128_t&  operator--
 int128_t   operator++
 int128_t   operator--
 int128_t   operator~
 int128_t   operator+

 // Math operators
 int128_t   operator+
 int128_t   operator-
 int128_t   operator*
 int128_t   operator/
 int128_t   operator%

 int128_t&  operator+=
 int128_t&  operator-=
 int128_t&  operator*=
 int128_t&  operator/=
 int128_t&  operator%=

 // Shift operators
 int128_t   operator>>
 int128_t   operator<<
 int128_t&  operator>>=
 int128_t&  operator<<=

 // Logical operators
 int128_t   operator^
 int128_t   operator|
 int128_t   operator&
 int128_t&  operator^=
 int128_t&  operator|=
 int128_t&  operator&=

 // Equality operators
 int        compare
 bool       operator==
 bool       operator!=
 bool       operator>
 bool       operator>=
 bool       operator<
 bool       operator<=

 // Operators to convert back to basic types
 int8_t     AsInt8
 int16_t    AsInt16
 int32_t    AsInt32
 int64_t    AsInt64
 float      AsFloat
 double     AsDouble

 // Misc. Functions
 void       Negate
 bool       IsNegative
 bool       IsPositive
 void       Modulus

 // String conversion functions
 int128_t   StrToInt128
 void       Int128ToStr
}

// Binary operators
int128_t
uint128_t   operator+
            operator-
            operator*
            operator/
            operator%
            operator^
            operator|
            operator&

int         compare
bool        operator==
bool        operator!=
bool        operator>
bool        operator>=
bool        operator<
bool        operator<=

const int128_t  INT128_MIN
const int128_t  INT128_MAX
const uint128_t UINT128_MIN
const uint128_t UINT128_MAX

#define INT128_C(x)
#define UINT128_C(x)
```