MinUtils Library Reference

version 0.13.0

Generated by Doxygen 1.8.12

Contents

1 Overview

MinUtils library is a part of MIN∗ library set that is obsolete and should be disbanded. Most of functionality goes to **MinBase** library.

The **MinUtils** library is header based, consisting of definitions, structures, classes with inline members, inline functions and templates, and as such do not need to be built in advance of its use.

The library is written in C++ and can be compiled under Linux (GCC) and Windows (MSVC 8 and later).

2 MinUtils License Agreements

2.1 Library License Agreement

MinUtils is released under FreeBSD License. It is free for both academic and commercial use.

2.2 Documentation License Agreement

This documentation is released under FreeBSD Documentation License. It is free for both academic and commercial use.

3 Module Documentation

3.1 Multi-channel Array Representation

The module specifies a multi-dimensional dense multi-channel array representation.

Data Structures

- struct MinArr

  A multi-dimensional dense multi-channel array representation. More...

3.1.1 Detailed Description

3.1.2 Data Structure Documentation

3.1.2.1 struct MinArr

The struct `MinArr` represents a multi-dimensional dense numerical single-channel or multi-channel array. The data layout of the array is defined by the field `MinArr::pStrides`. Let us M is an instance of `MinArr`. So the address of element $(i_0, ..., i_{M.dim-1})$, where $0 \leq i_k \leq M.pSizes[k]$ is computed as:

$$M.pStart + M.pStrides[0] * i_0 + ... + M.pStrides[M.dim - 1] * i_{M.dim-1}$$

For example, in the case of 2-dimensional array the above formula is reduced to:

$$M.pStart + M.pStrides[0] * i + M.pStrides[1] * j$$

Note that $M.pStrides[i] \geq M.pStrides[i+1]$ that is, 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by plane etc. $M.pStrides[M.dim - 1]$ is minimal and always equal to the element size $M.channelDepth$.

Definition at line 69 of file minarr.h.

Data Fields

- int32_t dim

  The number of array dimensions. It must be positive.
- int32_t * pSizes

  Array size of each dimension. It must be nonnegative.
- int32_t * pStrides

  Full row length (in bytes) for each dimension. It must be positive.
- int32_t channelDepth

  Size of element in bytes. It must be positive.
- MinFmt format

  Element format (supported formats are represented by MinFmt).
- uint8_t * pStart

  The pointer to the (0, 0, ... 0) element.

3.3 Mathematical operations

The module specifies mathematical operations which can be used in image processing functions. All operations are specified by the follow constants: OP_MIN (binary minimum), OP_MAX (binary maximum), OP_ADD (binary addition), OP_DIF (binary difference), OP_ADF (binary absolute difference), OP_MUL (binary multiplication), OP_AVE (binary average), and OP_EUC (binary Euclidean norm). Additionally, the operations are grouped into several enums in accordance with their properties: all binary operations (BiOp), associative operations (AsOp), commutative operations (CoOp), associative-commutative operations (AsCoOp) and idempotent operations (IdOp).

Enumerations

- enum MathOp

  Specifies mathematical operations.
- enum UnOp

  Specifies unary operations.
- enum BiOp

  Specifies binary operations.
- enum AsOp

  Specifies associative operations.
- enum CoOp

  Specifies commutative operations.
- enum AsCoOp

  Specifies associative-commutative operations.
- enum IdOp

  Specifies idempotent operations.

3.3.1 Detailed Description

3.3.2 Enumeration Type Documentation

3.3.2.1 MathOp

`enum MathOp`

The enum specifies mathematical operations.

Enumerator

OP_MIN	Specifies binary minimum operation. The constant specifies the binary minimum operation f that is defined as follows: $$f(x, y) = \min(x, y)$$
OP_MAX	Specifies binary maximum operation. The constant specifies the binary maximum operation f that is defined as follows: $$f(x, y) = \max(x, y)$$
OP_ADD	Specifies binary addition operation. The constant specifies the binary addition operation f that is defined as follows: $$f(x, y) = x + y$$

Enumerator

OP_DIF	Specifies binary difference operation. The constant specifies the binary difference operation f that is defined as follows: $$f(x,y) = x - y$$
OP_ADF	Specifies binary absolute difference operation. The constant specifies the binary absolute difference operation f that is defined as follows: $$f(x,y) =\mid x - y \mid$$
OP_MUL	Specifies binary multiplication operation. The constant specifies the binary multiplication operation f that is defined as follows: $$f(x,y) = x \cdot y$$
OP_AVE	Specifies binary average operation. The constant specifies the binary average operation f that is defined as follows: $$f(x,y) = \frac{x+y}{2}$$
OP_EUC	Specifies binary Euclidean norm operation. The constant specifies the binary Euclidean norm operation f that is defined as follows: $$f(x,y) = \sqrt{x^2+y^2}$$
OP_DIV	Specifies binary division operation. The constant specifies the binary multiplication operation f that is defined as follows: $$f(x,y) = x/y$$
OP_SSQ	Specifies binary sum of squares operation. The constant specifies the binary sum of squares operation f that is defined as follows: $$f(x,y) = x^2 + y^2$$
OP_ABS	Specifies unary absolute value operation. The constant specifies the unary absolute value operation f that is defined as follows: $$f(x) = \begin{cases} x & x >= 0 \\ -x & otherwise \end{cases}$$
OP_SQRT	Specifies unary square root operation. The constant specifies the unary square root operation f that is defined as follows: $$f(x) = \sqrt{x}$$
OP_POW	Specifies binary power operation. The constant specifies the binary power operation f that is defined as follows: $$f(x,y) = x^y$$
OP_INV	Specifies unary inversion operation. The constant specifies the unary inversion operation f that is defined as follows: $$f(x) =\sim x$$, where $\sim$ is bitwise not.

Enumerator

OP_AND	Specifies binary AND operation. The constant specifies the binary AND operation f that is defined as follows: $$f(x, y) = x \& y$$
OP_OR	Specifies binary OR operation. The constant specifies the binary OR operation f that is defined as follows: $$f(x, y) = x \mid y$$
OP_XOR	Specifies binary OR operation. The constant specifies the binary OR operation f that is defined as follows: $$f(x, y) = x \oplus y$$

Definition at line 64 of file mathoper.h.

3.3.2.2 UnOp

`enum UnOp`

The enum specifies unary operations, that is such ones which involve one operand. Formally, a unary operation f on a set S maps elements of S to S:

$$f : S \to S$$

Enumerator

UNOP_ABS	Unary operation for computing absolute value.
UNOP_SQRT	Unary operation for computing square root.
UNOP_INV	Unare operation for bitwise NOT.

Definition at line 194 of file mathoper.h.

3.3.2.3 BiOp

`enum BiOp`

The enum specifies binary operations, that is such ones which involve two operands. Formally, a binary operation f on a set S is a binary relation that maps elements of the Cartesian product $S \times S$ to S:

$$f : S \times S \to S$$

Enumerator

BIOP_MIN	Binary minimum operation (see OP_MIN).
BIOP_MAX	Binary maximum operation (see OP_MAX).
BIOP_ADD	Binary addition operation (see OP_ADD).
BIOP_DIF	Binary difference operation (see OP_DIF).
BIOP_ADF	Binary absolute difference operation (see OP_ADF).
BIOP_MUL	Binary multiplication operation (see OP_MUL).

Enumerator

BIOP_AVE	Binary average operation (see OP_AVE).
BIOP_EUC	Binary Euclidean norm operation (see OP_EUC).
BIOP_DIV	Binary divisiob operation (see OP_DIV).
BIOP_SSQ	Binary sum of squares operation (see OP_SSQ).
BIOP_POW	Binary power operation (see OP_POW).
BIOP_AND	Binary and operation (see OP_AND).
BIOP_OR	Binary or operation (see OP_OR).
BIOP_XOR	Binary xor operation (see OP_XOR).

Definition at line 209 of file mathoper.h.

3.3.2.4 AsOp

`enum AsOp`

The enum specifies associative operations that is such ones which can be freely regrouped without altering result. Formally, a binary operation f on a set S is called associative if it satisfies the associative law:

$$f(f(x,y),z) = f(x,f(y,z)) \quad \forall x,y,z \in S$$

Enumerator

ASOP_MIN	Binary minimum operation (see OP_MIN).
ASOP_MAX	Binary maximum operation (see OP_MAX).
ASOP_ADD	Binary addition operation (see OP_ADD).
ASOP_MUL	Binary multiplication operation (see OP_MUL).
ASOP_EUC	Binary Euclidean norm operation (see OP_EUC).
ASOP_AND	Binary and operation (see OP_AND).
ASOP_OR	Binary or operation (see OP_OR).
ASOP_XOR	Binary xor operation (see OP_XOR).

Definition at line 235 of file mathoper.h.

3.3.2.5 CoOp

`enum CoOp`

The enum specifies commutative operations, that is such ones which do not depend on the order of the input parameters. Formally, a binary operation f on a set S is called commutative if it satisfies the commutative law:

$$f(x,y) = f(y,x) \quad \forall x,y \in S$$

Enumerator

COOP_MIN	Binary minimum operation (see OP_MIN).
COOP_MAX	Binary maximum operation (see OP_MAX).
COOP_ADD	Binary addition operation (see OP_ADD).

Enumerator

COOP_ADF	Binary absolute difference operation (see OP_ADF).
COOP_MUL	Binary multiplication operation (see OP_MUL).
COOP_AVE	Binary average operation (see OP_AVE).
COOP_EUC	Binary Euclidean norm operation (see OP_EUC).
COOP_SSQ	Binary sum of squares operation (see OP_SSQ).
COOP_AND	Binary and operation (see OP_AND).
COOP_OR	Binary or operation (see OP_OR).
COOP_XOR	Binary xor operation (see OP_XOR).

Definition at line 255 of file mathoper.h.

3.3.2.6 AsCoOp

`enum AsCoOp`

The enum specifies associative-commutative operations, that is such ones which have both associative and commutative properties. Formally, a binary operation $\circ$ on a set S is called associative-commutative if it satisfies both the associative and the commutative laws:

$$f(f(x,y),z) = f(x,f(y,z)) \quad \forall x,y,z \in S$$

$$f(x,y) = f(y,x) \quad \forall x,y \in S$$

Enumerator

ASCOOP_MIN	Binary minimum operation (see OP_MIN).
ASCOOP_MAX	Binary maximum operation (see OP_MAX).
ASCOOP_ADD	Binary addition operation (see OP_ADD).
ASCOOP_MUL	Binary multiplication operation (see OP_MUL).
ASCOOP_EUC	Binary Euclidean norm operation (see OP_EUC).
ASCOOP_AND	Binary and operation (see OP_AND).
ASCOOP_OR	Binary or operation (see OP_OR).
ASCOOP_XOR	Binary xor operation (see OP_XOR).

Definition at line 280 of file mathoper.h.

3.3.2.7 IdOp

`enum IdOp`

The enum specifies idempotent operations, that is such ones which can be applied multiple times without changing the result. Formally, a binary operation f on a set S is called idempotent if

$$f(x,f(x,y)) = f(x,y) \quad \forall x,y \in S$$

Enumerator

IDOP_MIN	Binary minimum operation (see OP_MIN).
IDOP_MAX	Binary maximum operation (see OP_MAX).
IDOP_AND	Binary and operation (see OP_AND).
IDOP_OR	Binary or operation (see OP_OR).

Definition at line 299 of file mathoper.h.

3.4 Smart Pointers

The module specifies classes which will take care about freeing memory of allocated array when the scope of such array ends.

Data Structures

- class scoped_c_array

  Specifies a class which will take care about freeing memory with `free()` function. More...
- class scoped_cpp_array

  Specifies a class which will take care about freeing memory with `delete[]` function. More...

Macros

- #define DEFINE_SCOPED_OBJECT(name, freeing_proc)

  Defines a template class which will take care about freeing memory of allocated array when the scope of such array ends.

3.4.1 Detailed Description

3.4.2 Data Structure Documentation

3.4.2.1 class scoped_c_array

The class takes care about freeing memory with `free()` function.

3.4.2.2 class scoped_cpp_array

The class takes care about freeing memory with `delete`[] function.

3.4.3 Macro Definition Documentation

3.4.3.1 DEFINE_SCOPED_OBJECT

```
#define DEFINE_SCOPED_OBJECT(
                name,
                freeing_proc )
```

Value:

```
template<typename T> class name \
{ \
public: \
  explicit name(T *p = 0): p(p) {} \
  ~name() \
  { \
    freeing_proc(p); \
  } \
  operator T *() const \
  { \
    return p; \
  } \
private: \
  name(const name &); \
  void operator =(const name &); \
  T *p; \
};
```

There are at least two types of arrays: allocated by new[] call (see `scoped_cpp_array` class) and allocated by malloc-like call (see `scoped_c_array` class). The code of classes for both cases is the same but one line in destructor. So the macro is defined for that purpose.

Definition at line 62 of file smartptr.h.

3.5 C99 Standard Data Types

The module describes the `stdint.h` file with is a header file in the C standard library introduced in the C99 standard library section 7.18 to allow programmers to write more portable code by providing a set of typedefs that specify exact-width integer types, together with the defined minimum and maximum allowable values for each type, using macros. This header is particularly useful for embedded programming which often involves considerable manipulation of hardware specific I/O registers requiring integer data of fixed widths, specific locations and exact alignments.

Macros

- #define _W64

  Defines `_W64` macros to mark types changing their size, like `intptr_t` or `uintptr_t`.
- #define INT8_MIN ((int8_t)_I8_MIN)

  Defines a minimum value of a signed 8-bit integer.
- #define INT8_MAX _I8_MAX

  Defines a maximum value of a signed 8-bit integer.
- #define INT16_MIN ((int16_t)_I16_MIN)

  Defines a minimum value of a signed 16-bit integer.
- #define INT16_MAX _I16_MAX

  Defines a maximum value of a signed 16-bit integer.
- #define INT32_MIN ((int32_t)_I32_MIN)

  Defines a minimum value of a signed 32-bit integer.
- #define INT32_MAX _I32_MAX

  Defines a maximum value of a signed 32-bit integer.
- #define INT64_MIN ((int64_t)_I64_MIN)

  Defines a minimum value of a signed 64-bit integer.
- #define INT64_MAX _I64_MAX

  Defines a maximum value of a signed 64-bit integer.
- #define UINT8_MAX _UI8_MAX

  Defines a maximum value of an unsigned 8-bit integer.
- #define UINT16_MAX _UI16_MAX

  Defines a maximum value of an unsigned 16-bit integer.
- #define UINT32_MAX _UI32_MAX

  Defines a maximum value of an unsigned 32-bit integer.
- #define UINT64_MAX _UI64_MAX

  Defines a maximum value of an unsigned 64-bit integer.
- #define INT_LEAST8_MIN INT8_MIN

  Defines a minimum value of a signed integer with a width of at least 8 bits.
- #define INT_LEAST8_MAX INT8_MAX

  Defines a maximum value of a signed integer with a width of at least 8 bits.
- #define INT_LEAST16_MIN INT16_MIN

  Defines a minimum value of a signed integer with a width of at least 16 bits.
- #define INT_LEAST16_MAX INT16_MAX

  Defines a maximum value of a signed integer with a width of at least 16 bits.
- #define INT_LEAST32_MIN INT32_MIN

  Defines a minimum value of a signed integer with a width of at least 32 bits.
- #define INT_LEAST32_MAX INT32_MAX

  Defines a maximum value of a signed integer with a width of at least 32 bits.
- #define INT_LEAST64_MIN INT64_MIN

Defines a minimum value of a signed integer with a width of at least 64 bits.

- #define INT_LEAST64_MAX INT64_MAX

  Defines a maximum value of a signed integer with a width of at least 64 bits.
- #define UINT_LEAST8_MAX UINT8_MAX

  Defines a maximum value of an unsigned integer with a width of at least 8 bits.
- #define UINT_LEAST16_MAX UINT16_MAX

  Defines a maximum value of an unsigned integer with a width of at least 16 bits.
- #define UINT_LEAST32_MAX UINT32_MAX

  Defines a maximum value of an unsigned integer with a width of at least 32 bits.
- #define UINT_LEAST64_MAX UINT64_MAX

  Defines a maximum value of an unsigned integer with a width of at least 64 bits.
- #define INT_FAST8_MIN INT8_MIN

  Defines a minimum value of a fastest signed 8-bit integer.
- #define INT_FAST8_MAX INT8_MAX

  Defines a maximum value of a fastest signed 8-bit integer.
- #define INT_FAST16_MIN INT16_MIN

  Defines a minimum value of a fastest signed 16-bit integer.
- #define INT_FAST16_MAX INT16_MAX

  Defines a maximum value of a fastest signed 16-bit integer.
- #define INT_FAST32_MIN INT32_MIN

  Defines a minimum value of a fastest signed 32-bit integer.
- #define INT_FAST32_MAX INT32_MAX

  Defines a maximum value of a fastest signed 32-bit integer.
- #define INT_FAST64_MIN INT64_MIN

  Defines a minimum value of a fastest signed 64-bit integer.
- #define INT_FAST64_MAX INT64_MAX

  Defines a maximum value of a fastest signed 64-bit integer.
- #define UINT_FAST8_MAX UINT8_MAX

  Defines a maximum value of a fastest unsigned 8-bit integer.
- #define UINT_FAST16_MAX UINT16_MAX

  Defines a maximum value of a fastest unsigned 16-bit integer.
- #define UINT_FAST32_MAX UINT32_MAX

  Defines a maximum value of a fastest unsigned 32-bit integer.
- #define UINT_FAST64_MAX UINT64_MAX

  Defines a maximum value of a fastest unsigned 64-bit integer.
- #define INTPTR_MIN INT32_MIN

  Defines a minimum value of a signed integer which is guaranteed to hold the value of a pointer.
- #define INTPTR_MAX INT32_MAX

  Defines a maximum value of a signed integer which is guaranteed to hold the value of a pointer.
- #define UINTPTR_MAX UINT32_MAX

  Defines a maximum value of an unsigned integer which is guaranteed to hold the value of a pointer.
- #define INTMAX_MIN INT64_MIN

  Defines a minimum value of a signed integer which has the greatest limits.
- #define INTMAX_MAX INT64_MAX

  Defines a maximum value of a signed integer which has the greatest limits.
- #define UINTMAX_MAX UINT64_MAX

  Defines a maximum value of an unsigned integer which has the greatest limits.
- #define PTRDIFF_MIN _I32_MIN

  Defines a minimum value `ptrdiff_t` can hold.
- #define PTRDIFF_MAX _I32_MAX

  Defines a maximum value `ptrdiff_t` can hold.

- #define SIG_ATOMIC_MIN INT_MIN

  Defines a minimum value `sig_atomic_t` can hold.
- #define SIG_ATOMIC_MAX INT_MAX

  Defines a maximum value `sig_atomic_t` can hold.
- #define SIZE_MAX _UI32_MAX

  Defines a maximum value `size_t` can hold.
- #define WCHAR_MIN 0

  Defines a minimum value for type `wchar_t`.
- #define WCHAR_MAX _UI16_MAX

  Defines a maximum value for type `wchar_t`.
- #define WINT_MIN 0

  Defines a minimum value for type `wint_t`.
- #define WINT_MAX _UI16_MAX

  Defines a maximum value for type `wint_t`.
- #define INT8_C(val) val##i8

  Defines a macros which converts an integer literal to a signed integer with a width of at least 8 bits.
- #define INT16_C(val) val##i16

  Defines a macros which converts an integer literal to a signed integer with a width of at least 16 bits.
- #define INT32_C(val) val##i32

  Defines a macros which converts an integer literal to a signed integer with a width of at least 32 bits.
- #define INT64_C(val) val##i64

  Defines a macros which converts an integer literal to a signed integer with a width of at least 64 bits.
- #define UINT8_C(val) val##ui8

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 8 bits.
- #define UINT16_C(val) val##ui16

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 16 bits.
- #define UINT32_C(val) val##ui32

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 32 bits.
- #define UINT64_C(val) val##ui64

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 64 bits.
- #define INTMAX_C INT64_C

  Defines a macros which converts an integer literal to a signed integer which has the greatest limits.
- #define UINTMAX_C UINT64_C

  Defines a macros which converts an integer literal to an unsigned integer which has the greatest limits.

Typedefs

- typedef signed char int8_t

  Defines a signed integer type with a width of exactly 8 bits.
- typedef signed short int16_t

  Defines a signed integer type with a width of exactly 16 bits.
- typedef signed int int32_t

  Defines a signed integer type with a width of exactly 32 bits.
- typedef signed __int64 int64_t

  Defines a signed integer type with a width of exactly 64 bits.
- typedef unsigned char uint8_t

  Defines an unsigned integer type with a width of exactly 8 bits.
- typedef unsigned short uint16_t

  Defines an unsigned integer type with a width of exactly 16 bits.
- typedef unsigned int uint32_t

Defines an unsigned integer type with a width of exactly 32 bits.

- typedef unsigned __int64 uint64_t

  Defines an unsigned integer type with a width of exactly 64 bits.
- typedef int8_t int_least8_t

  Defines a signed integer type with a width of at least 8 bits.
- typedef int16_t int_least16_t

  Defines a signed integer type with a width of at least 16 bits.
- typedef int32_t int_least32_t

  Defines a signed integer type with a width of at least 32 bits.
- typedef int64_t int_least64_t

  Defines a signed integer type with a width of at least 64 bits.
- typedef uint8_t uint_least8_t

  Defines an unsigned integer type with a width of at least 8 bits.
- typedef uint16_t uint_least16_t

  Defines an unsigned integer type with a width of at least 16 bits.
- typedef uint32_t uint_least32_t

  Defines an unsigned integer type with a width of at least 32 bits.
- typedef uint64_t uint_least64_t

  Defines an unsigned integer type with a width of at least 64 bits.
- typedef int8_t int_fast8_t

  Defines a signed integer type being usually fastest with a width of at least 8 bits.
- typedef int16_t int_fast16_t

  Defines a signed integer type being usually fastest with a width of at least 16 bits.
- typedef int32_t int_fast32_t

  Defines a signed integer type being usually fastest with a width of at least 32 bits.
- typedef int64_t int_fast64_t

  Defines a signed integer type being usually fastest with a width of at least 64 bits.
- typedef uint8_t uint_fast8_t

  Defines an unsigned integer type being usually fastest with a width of at least 8 bits.
- typedef uint16_t uint_fast16_t

  Defines an unsigned integer type being usually fastest with a width of at least 16 bits.
- typedef uint32_t uint_fast32_t

  Defines an unsigned integer type being usually fastest with a width of at least 32 bits.
- typedef uint64_t uint_fast64_t

  Defines an unsigned integer type being usually fastest with a width of at least 64 bits.
- typedef _W64 signed int intptr_t

  Defines a signed integer type which is guaranteed to hold the value of a pointer.
- typedef _W64 unsigned int uintptr_t

  Defines an unsigned integer type which is guaranteed to hold the value of a pointer.
- typedef int64_t intmax_t

  Defines a signed integer type which has the greatest limits.
- typedef uint64_t uintmax_t

  Defines an unsigned integer type which has the greatest limits.

3.5.1 Detailed Description

4 Data Structure Documentation

4.1 se::CommonValue Class Reference

Public Types

- enum **Type**

Public Member Functions

- **CommonValue** (int32_t vl)
- **CommonValue** (real64_t vl)
- **CommonValue** (const char *vl)
- **CommonValue** (bool vl)
- **CommonValue** (const CommonValue &cv)
- CommonValue & **operator=** (const CommonValue &r)
- int32_t **to_int** () const
- real64_t **to_real** () const
- const char * **to_str** () const
- bool **to_bool** () const
- Type **valType** ()
- bool **empty** ()

Protected Attributes

- Type **tp**
- int32_t **vlInt**
- real64_t **vlReal**
- std::string **vlStr**
- bool **vlBool**

Friends

- bool **operator==** (const CommonValue &l, const CommonValue &r)
- bool **operator!=** (const CommonValue &l, const CommonValue &r)
- CommonValue **from_str** (const char *vl)

4.1.1 Detailed Description

Definition at line 121 of file listfile.h.

4.2 LifeTime::Entry Struct Reference

Friends

- class **LifeTime**

4.2.1 Detailed Description

Definition at line 20 of file lifetime.h.

4.3 se::GenUniqueInt Class Reference

Public Member Functions

- **GenUniqueInt** (int startFrom=0)
- int **generate** ()
- int **getLast** ()

Private Attributes

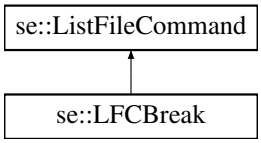
- int **curNum**

4.3.1 Detailed Description

Definition at line 109 of file listfile.h.

4.4 se::LFCBreak Class Reference

Inheritance diagram for se::LFCBreak:



Public Member Functions

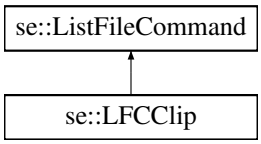
- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- virtual void **writeToConsole** (proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const

4.4.1 Detailed Description

Definition at line 549 of file listfile.h.

4.5 se::LFCClip Class Reference

Inheritance diagram for se::LFCClip:



Public Member Functions

- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- virtual void **writeToConsole** (proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const
- void **parseCmdParams** (const char *pszCmdParams)

Protected Attributes

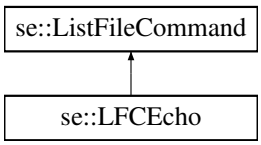
- pathname **fileName**

4.5.1 Detailed Description

Definition at line 625 of file listfile.h.

4.6 se::LFCEcho Class Reference

Inheritance diagram for se::LFCEcho:



Public Member Functions

- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const
- void **parseCmdParams** (const char *pszCmdParams)

Protected Attributes

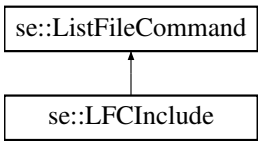
- std::string **sText**

4.6.1 Detailed Description

Definition at line 642 of file listfile.h.

4.7 se::LFCInclude Class Reference

Inheritance diagram for se::LFCInclude:



Public Member Functions

- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- virtual void **writeToConsole** (proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const
- virtual void **parseCmdString** (const char *pszCmdString)
- void **parseCmdParams** (const char *pszCmdParams)

Protected Attributes

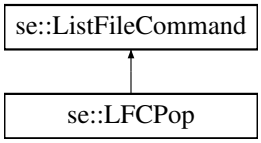
- pathname **fileName**

4.7.1 Detailed Description

Definition at line 602 of file listfile.h.

4.8 se::LFCPop Class Reference

Inheritance diagram for se::LFCPop:



Public Member Functions

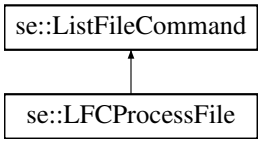
- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const

4.8.1 Detailed Description

Definition at line 723 of file listfile.h.

4.9 se::LFCProcessFile Class Reference

Inheritance diagram for se::LFCProcessFile:



Public Member Functions

- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- virtual void **writeToConsole** (proc_list_data &pld) const
- LFCProcessFile * **construct** () const
- const char * **statement** () const
- void **parseCmdParams** (const char *pszCmdParams)

Protected Attributes

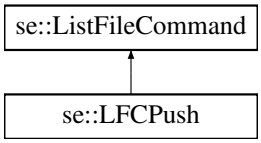
- pathname **fileName**

4.9.1 Detailed Description

Definition at line 516 of file listfile.h.

4.10 se::LFCPush Class Reference

Inheritance diagram for se::LFCPush:



Public Member Functions

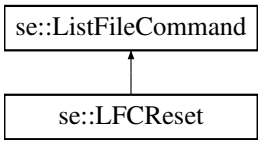
- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const

4.10.1 Detailed Description

Definition at line 706 of file listfile.h.

4.11 se::LFCReset Class Reference

Inheritance diagram for se::LFCReset:



Public Member Functions

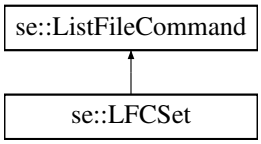
- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const

4.11.1 Detailed Description

Definition at line 694 of file listfile.h.

4.12 se::LFCSet Class Reference

Inheritance diagram for se::LFCSet:



Public Member Functions

- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const
- void **parseCmdParams** (const char *pszCmdParams)

Protected Attributes

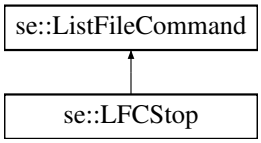
- VarSetT **variables**

4.12.1 Detailed Description

Definition at line 659 of file listfile.h.

4.13 se::LFCStop Class Reference

Inheritance diagram for se::LFCStop:



Public Member Functions

- int **execute** (ProcFile &procFile, proc_list_data &pld) const
- virtual void **writeToConsole** (proc_list_data &pld) const
- ListFileCommand * **construct** () const
- const char * **statement** () const

4.13.1 Detailed Description

Definition at line 535 of file listfile.h.

4.14 LifeTime Class Reference

Data Structures

- struct Entry

Public Member Functions

- void **put** (Entry *pEntry)
- void **kill** (Entry *pEntry)
- void **killThemAll** ()
- void **release** (Entry *pEntry)

Private Member Functions

- **LifeTime** (const LifeTime &)
- LifeTime & **operator=** (const LifeTime &)

Private Attributes

- std::set< Entry * > **m_entries**

Friends

- class **LifeTimeUtils**

4.14.1 Detailed Description

Definition at line 11 of file lifetime.h.

4.15 LifeTimeUtils Class Reference

Static Public Member Functions

- static void **move** (LifeTime &destination, LifeTime &source)

4.15.1 Detailed Description

Definition at line 100 of file lifetime.h.

4.16 se::listfile Class Reference

Public Member Functions

- **listfile** (const char *listfilename)
- void **registerCommands** ()
- int **size** ()
- const ListFileCommand * **operator[ ]** (int i)

Private Types

- typedef std::map< std::string, ListFileCommand * > **CommandRegistry**

Private Attributes

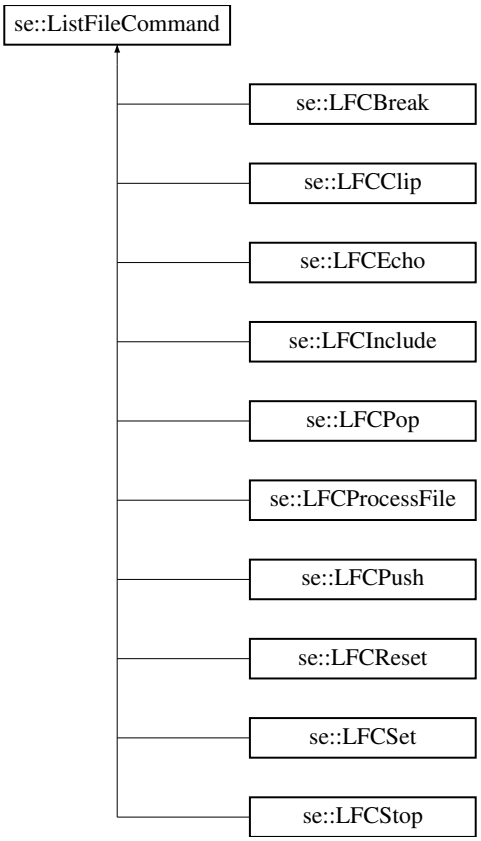
- std::vector< ListFileCommand * > **commandList**
- CommandRegistry **commandRegistry**

4.16.1 Detailed Description

Definition at line 735 of file listfile.h.

4.17 se::ListFileCommand Class Reference

Inheritance diagram for se::ListFileCommand:



Public Member Functions

- virtual int **execute** (ProcFile &procFile, proc_list_data &pld) const =0
- virtual void **writeToConsole** (proc_list_data &pld) const
- virtual const char * **statement** () const =0
- virtual ListFileCommand * **construct** () const =0
- void **preprocAndParseCmdString** (const char *pszCmdString)
- virtual void **parseCmdString** (const char *pszCmdString)
- ListFileCommand * **constructFromString** (const char *pszCmdString) const
- virtual void **parseCmdParams** (const char *pszCmdParams)

4.17.1 Detailed Description

Definition at line 490 of file listfile.h.

4.18 se::pathname Class Reference

Public Member Functions

- **pathname** (const char *str=NULL)
- void **parse** (const char *str=NULL)
- bool **is_relative** () const

Data Fields

- std::string **path**
- std::string **folder**
- std::string **name**
- std::string **name_base**
- std::string **ext**

4.18.1 Detailed Description

Definition at line 46 of file pathname.h.

4.19 se::proc_list_data Class Reference

Public Member Functions

- **proc_list_data** (int _flags=0)
- **proc_list_data** (const proc_list_data &parentPLD, const char *pszImgFolder, const char *pszLstPathName)

Data Fields

- int **files_ok**
- int **files_failed**
- double **last_file_seconds**
- double **list_seconds**
- int **total_files_ok**
- int **total_files_failed**
- double **total_seconds**
- int **cur_list_done**
- int **cur_list_total**
- std::string **margin**
- std::string **sImgRootFolder**
- std::string **sListFilePathName**
- std::string **sReportRootFolder**
- std::string **sSubfolder**
- std::string **sIdealRootFolder**
- bool **bStopped**
- bool **bBreak**
- std::vector< VarSetT > **variables**
- GenUniqueInt * **pClipNumGenerator**
- bool **bOwnClipNumGenerator**
- std::ostream * **pOsFileListReport**
- std::vector< std::string > * **pFileListContainer**
- bool **bDoNotRecog**
- int **flags**

4.19.1 Detailed Description

Definition at line 325 of file listfile.h.

4.20 se::ProcFile Class Reference

Data Structures

- class RunParams

Public Member Functions

- int **processFile** (proc_list_data &pld, const pathname &fileName)
- virtual int **run** (const RunParams &params)
- virtual int **finish** (const VarSetT &vars, const char *pszLstPath)
- int **runAndReport** (const RunParams &params, std::ostream *pOs, std::vector< std::string > *pListFile↩ Container, bool bDoNotRecog)

4.20.1 Detailed Description

Definition at line 401 of file listfile.h.

4.21 ProcImageResult Class Reference

Public Member Functions

- **ProcImageResult** (const std::string &name="", const std::string &_info="")
- **ProcImageResult** (const ProcImageResult &other)
- ProcImageResult & **operator=** (const ProcImageResult &other)

Data Fields

- std::string **jpgName**
- std::string **info**
- Residuals **diffWithIdeal**
- TimeProfile **timeProfile**

4.21.1 Detailed Description

Definition at line 54 of file report.h.

4.22 Residuals Class Reference

Public Member Functions

- **Residuals** (const Residuals &other)
- Residuals & **operator=** (const Residuals &other)
- void **setValue** (const char *key, double value)
- double **getValue** (const char *key) const
- double **getMax** (const std::set< std::string > &subsystems) const
- void **updateSubsystems** (std::set< std::string > &subsystems, const std::string &pattern="") const
- bool **isEmpty** () const
- bool **write** (std::fstream &out) const
- bool **read** (std::fstream &in)

Private Attributes

- std::map< std::string, double > **values**

Static Private Attributes

- static const int **unknownValue** = -1

4.22.1 Detailed Description

Definition at line 45 of file residuals.h.

4.23 se::ProcFile::RunParams Class Reference

Public Member Functions

- **RunParams** (const VarSetT &v)

Data Fields

- const char * **pszFilePathName**
- const char * **pszSymmetricPathFromLst**
- const char * **pszIdealFolder**
- const char * **pszReportRoot**
- const char * **pszLstFolder**
- const char * **pszLstPathName**
- const char * **pszRelativePath**
- int **flags**
- const VarSetT & **vars**

4.23.1 Detailed Description

Definition at line 404 of file listfile.h.

4.24 Timer::Time Struct Reference

Data Fields

- int **unused**

4.24.1 Detailed Description

Definition at line 25 of file timer.h.

4.25 TimeProfile Class Reference

Public Member Functions

- **TimeProfile** (const TimeProfile &other)
- TimeProfile & **operator=** (const TimeProfile &other)
- void **setValue** (const char *key, double value)
- double **getValue** (const char *key) const
- void **updateSubsystems** (std::set< std::string > &subsystems, const std::string &pattern="") const
- bool **isEmpty** () const
- bool **write** (std::fstream &out) const
- bool **read** (std::fstream &in)

Private Attributes

- std::map< std::string, double > **values**

Static Private Attributes

- static const int **unknownValue** = 0

4.25.1 Detailed Description

Definition at line 45 of file timeprofile.h.

4.26 Timer Class Reference

Data Structures

- struct Time

Public Member Functions

- **Timer** (std::string name, TimeProfile *profile)
- void **start** ()
- long time ()

Private Member Functions

- **Timer** (const Timer &)
- Timer & **operator=** (const Timer &)
- **Timer** (const Timer &)
- Timer & **operator=** (const Timer &)
- void **getCurrentTime** (Time &t)
- long **getTimeDiff** (const Time &begin, const Time &end)

Private Attributes

- std::string **timerName**
- long long **t_start**
- TimeProfile * **profile**
- Time **m_start**

4.26.1 Detailed Description

Definition at line 129 of file timeprofile.h.

4.26.2 Member Function Documentation

4.26.2.1 time()

```
long Timer::time ( )  [inline]
```

Retrives time passed from start() in milliseconds

Definition at line 62 of file timer.h.

5 File Documentation

5.1 listfile.h File Reference

listfile (.lst) support and batch processing template

Data Structures

- class se::GenUniqueInt
- class se::CommonValue
- class se::proc_list_data
- class se::ProcFile
- class se::ProcFile::RunParams
- class se::ListFileCommand
- class se::LFCProcessFile
- class se::LFCStop
- class se::LFCBreak
- class se::LFCInclude
- class se::LFCClip
- class se::LFCEcho
- class se::LFCSet
- class se::LFCReset
- class se::LFCPush
- class se::LFCPop
- class se::listfile

Macros

- #define **MINUTILS_LISTFILE_H_INCLUDED**
- #define **LISTFILE_VERSION** 5

Typedefs

- typedef std::map< std::string, CommonValue > **se::VarSetT**

Functions

- std::string **se::substrex** (std::string s, std::string::size_type _Off=0, std::string::size_type _Count=std::string::npos)
- std::string **se::joinPath** (const std::string &sPart1, const std::string &sPart2)
- std::string **se::joinPath** (const std::string &sPart1, const std::string &sPart2, const std::string &sPart3)
- std::string **se::joinPath** (const std::string &sPart1, const std::string &sPart2, const std::string &sPart3, const std::string &sPart4)
- int32_t **se::roundReal** (real64_t val)
- CommonValue **se::from_str** (const char *vl)
- int **se::extractIntVariable** (const VarSetT &varset, const char *pszName, int32_t defaultValue)
- real64_t **se::extractRealVariable** (const VarSetT &varset, const char *pszName, real64_t defaultValue)
- int **se::extractBoolVariable** (const VarSetT &varset, const char *pszName, bool defaultValue)
- const char * **se::extractStrVariable** (const VarSetT &varset, const char *pszName, const char *pszDefaultValue)
- std::string **se::preprocCmdString** (const char *pszCmdString)
- int **se::proc_list_file** (const char *listfile_or_terminal_file_name, proc_list_data &pld, ProcFile *p_procfile)

Variables

- const char **se::CLIPNUMBER_VAR_NAME** [ ] = "clipNumber"
- const int **se::plf_verbose** = 0x001
- const int **se::plf_stop_if_result_less_zero** = 0x002

5.2 listfile.h

```
00001 /*
00002
00003 Copyright (c) 2011, Smart Engines Limited. All rights reserved.
00004
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without modification,
00008 are permitted provided that the following conditions are met:
00009
00010    1. Redistributions of source code must retain the above copyright notice,
00011       this list of conditions and the following disclaimer.
00012
00013    2. Redistributions in binary form must reproduce the above copyright notice,
00014       this list of conditions and the following disclaimer in the documentation
00015       and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00020 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
00027
00028 The views and conclusions contained in the software and documentation are those
00029 of the authors and should not be interpreted as representing official policies,
00030 either expressed or implied, of copyright holders.
00031
00032 */
00033
00039 #pragma once
00040 #ifndef MINUTILS_LISTFILE_H_INCLUDED
00041 #define MINUTILS_LISTFILE_H_INCLUDED
00042
00043 #define LISTFILE_VERSION 5
00044
00045 #include <vector>
00046 #include <string>
00047 #include <cstring>
00048 #include <map>
00049 #include <minutils/pathname.h>
00050 #include <minstopwatch/hiresclock.h>
00051 #include <minbase/mintyp.h>
00052 #include <minbase/minresult.h>
00053
00054 #include <iostream>
00055 #include <fstream>
00056 #include <sstream>
00057
00058 namespace se {
00059
00060 inline std::string substrex(
00061     std::string s,
00062     std::string::size_type _Off = 0,
00063     std::string::size_type _Count = std::string::npos)
00064 {
00065   if (_Off == std::string::npos)
00066     return std::string();
00067   return s.substr(_Off, _Count);
00068 }
00069
00070 inline
00071 std::string joinPath(
00072   const std::string &sPart1,
00073   const std::string &sPart2)
00074 {
00075   std::string sRes = sPart1;
00076   if (!sRes.empty()
00077     && (sRes[sRes.length() - 1] != '/')
00078     && (sRes[sRes.length() - 1] != '\\')
00079     && !sPart2.empty()
00080   )
00081     sRes += "/";
00082   sRes += sPart2;
00083   return sRes;
00084 }
00085
00086 inline
00087 std::string joinPath(
00088   const std::string &sPart1,
00089   const std::string &sPart2,
00090   const std::string &sPart3)
00091 {
00092   return joinPath(joinPath(sPart1, sPart2), sPart3);
00093 }
00094
00095 inline
00096 std::string joinPath(
00097   const std::string &sPart1,
00098   const std::string &sPart2,
00099   const std::string &sPart3,
00100   const std::string &sPart4)
00101 {
00102   return joinPath(joinPath(sPart1, sPart2, sPart3), sPart4);
00103 }
00104
00105 inline int32_t roundReal(real64_t val) {
00106   return static_cast<int32_t>((val > 0) ? (val + .5) : (val - .5));
00107 }
00108
00109 class GenUniqueInt
00110 {
00111 public:
00112   GenUniqueInt(int startFrom = 0)
00113     : curNum(startFrom - 1)
00114   {}
00115   int generate() { return ++curNum; }
00116   int getLast() { return curNum; }
00117 private:
00118   int curNum;
```

```
00119 };
00120
00121 class CommonValue
00122 {
00123 public:
00124   enum Type {TP_DEFAULT, TP_INT, TP_REAL, TP_STR, TP_BOOL};
00125
00126   CommonValue()
00127     : tp(TP_DEFAULT), vlInt(0), vlReal(0), vlBool(false)
00128   {}
00129   CommonValue(int32_t vl)
00130     : tp(TP_INT), vlInt(vl), vlReal(0), vlBool(false)
00131   {}
00132   CommonValue(real64_t vl)
00133     : tp(TP_REAL), vlInt(0), vlReal(vl), vlBool(false)
00134   {}
00135   CommonValue(const char *vl)
00136     : tp(TP_STR), vlInt(0), vlReal(0), vlStr(vl), vlBool(false)
00137   {}
00138   CommonValue(bool vl)
00139     : tp(TP_BOOL), vlInt(0), vlReal(0), vlBool(vl)
00140   {}
00141   CommonValue(const CommonValue &cv)
00142     : tp(cv.tp), vlInt(cv.vlInt), vlReal(cv.vlReal), vlStr(cv.vlStr), vlBool(cv.vlBool)
00143   {}
00144   CommonValue &operator=(const CommonValue &r) {
00145     tp = r.tp;
00146     vlInt = r.vlInt;
00147     vlReal = r.vlReal;
00148     vlStr = r.vlStr;
00149     vlBool = r.vlBool;
00150     return *this;
00151   }
00152   friend bool operator==(const CommonValue &l, const CommonValue &r) {
00153     if (l.tp != r.tp)
00154       return false;
00155     if (l.tp == TP_INT)
00156       return l.vlInt == r.vlInt;
00157     if (l.tp == TP_REAL)
00158       return l.vlReal == r.vlReal;
00159     if (l.tp == TP_BOOL)
00160       return l.vlBool == r.vlBool;
00161     if (l.tp == TP_STR)
00162       return l.vlStr == r.vlStr;
00163     if (l.tp == TP_DEFAULT)
00164       return true;
00165     return false;
00166   }
00167   friend bool operator!=(const CommonValue &l, const CommonValue &r) {
00168     return !(l == r);
00169   }
00170
00171   friend CommonValue from_str(const char *vl);
00172   int32_t to_int() const {
00173     switch(tp) {
00174       case TP_INT:
00175         return vlInt;
00176       case TP_REAL:
00177         return roundReal(vlReal);
00178       case TP_BOOL:
00179         return static_cast<int32_t>(vlBool);
00180       default:
00181         return 0;
00182     }
00183   }
00184   real64_t to_real() const {
00185     switch(tp) {
00186       case TP_INT:
00187         return vlInt;
00188       case TP_REAL:
00189         return vlReal;
00190       case TP_BOOL:
00191         return vlBool ? 1. : 0;
00192       default:
00193         return 0;
00194     }
00195     return 0;
00196   }
00197   const char *to_str() const {
00198     std::ostringstream oss;
00199     switch(tp) {
00200       case TP_INT:
00201         oss << vlInt;
00202         const_cast<CommonValue*>(this)->vlStr = oss.str();
00203         return vlStr.c_str();
00204       case TP_REAL:
00205         oss << vlReal;
```

```
00206         const_cast<CommonValue*>(this)->vlStr = oss.str();
00207         return vlStr.c_str();
00208       case TP_STR:
00209         return vlStr.c_str();
00210       default:
00211         const_cast<CommonValue*>(this)->vlStr = std::string();
00212         return vlStr.c_str();
00213     }
00214     return vlStr.c_str();
00215   }
00216   bool to_bool() const {
00217     switch(tp) {
00218       case TP_INT:
00219         return vlInt != 0;
00220       case TP_REAL:
00221         return vlReal != 0;
00222       case TP_STR:
00223         return false;
00224       case TP_BOOL:
00225         return vlBool;
00226       default:
00227         return false;
00228     }
00229     return false;
00230   }
00231   Type valType() { return tp; }
00232   bool empty() { return tp == TP_DEFAULT; }
00233 protected:
00234   Type tp;
00235   int32_t vlInt;
00236   real64_t vlReal;
00237   std::string vlStr;
00238   bool vlBool;
00239 };
00240
00241 inline
00242 CommonValue from_str(const char *vl) {
00243   CommonValue cv;
00244   if (!vl || !*vl)
00245     return cv;
00246   using std::string;
00247   if (!strcmp(vl, "false")) {
00248     cv.tp = CommonValue::TP_BOOL;
00249     cv.vlBool = false;
00250   } else if (!strcmp(vl, "true")) {
00251     cv.tp = CommonValue::TP_BOOL;
00252     cv.vlBool = true;
00253   } else if (vl[0] == '"')
00254   {
00255     cv.tp = CommonValue::TP_STR;
00256     cv.vlStr = vl + 1;
00257     if (cv.vlStr[cv.vlStr.length() - 1] == '"')
00258       cv.vlStr.resize(cv.vlStr.length() - 1);
00259   } else
00260   {
00261     string sVal(vl);
00262     if (sVal.find_first_not_of("0123456789+-.") == string::npos)
00263     {
00264       std::istringstream iss(sVal);
00265       if (sVal.find_first_of('.') != string::npos)
00266       {
00267         cv.tp = CommonValue::TP_REAL;
00268         iss >> cv.vlReal;
00269       } else
00270       {
00271         cv.tp = CommonValue::TP_INT;
00272         iss >> cv.vlInt;
00273       }
00274     } else
00275     {
00276       cv.tp = CommonValue::TP_STR;
00277       cv.vlStr = sVal;
00278     }
00279   }
00280   return cv;
00281 }
00282
00283 typedef std::map<std::string, CommonValue> VarSetT;
00284
00285 inline
00286 int extractIntVariable(const VarSetT &varset, const char *pszName, int32_t defaultValue)
00287 {
00288   VarSetT::const_iterator it = varset.find(pszName);
00289   if (it != varset.end())
00290     return it->second.to_int();
00291   else
00292     return defaultValue;
```

```
00293 }
00294
00295 inline
00296 real64_t extractRealVariable(const VarSetT &varset, const char *pszName, real64_t defaultValue)
00297 {
00298   VarSetT::const_iterator it = varset.find(pszName);
00299   if (it != varset.end())
00300     return it->second.to_real();
00301   else
00302     return defaultValue;
00303 }
00304
00305 inline
00306 int extractBoolVariable(const VarSetT &varset, const char *pszName, bool defaultValue)
00307 {
00308   VarSetT::const_iterator it = varset.find(pszName);
00309   if (it != varset.end())
00310     return it->second.to_bool();
00311   else
00312     return defaultValue;
00313 }
00314
00315 inline
00316 const char *extractStrVariable(const VarSetT &varset, const char *pszName, const char *pszDefaultValue)
00317 {
00318   VarSetT::const_iterator it = varset.find(pszName);
00319   if (it != varset.end())
00320     return it->second.to_str();
00321   else
00322     return pszDefaultValue;
00323 }
00324
00325 class proc_list_data
00326 {
00327 public:
00328   int files_ok;
00329   int files_failed;
00330   double last_file_seconds;
00331   double list_seconds;
00332
00333   int total_files_ok;     //
00334   int total_files_failed; //  with included
00335   double total_seconds;   //
00336
00337   int cur_list_done; // [done/total] to show progress on current list
00338   int cur_list_total; // [1/156]...[156/156] == [done/total]
00339
00340   std::string margin; // for nested list output formatting
00341
00342   std::string sImgRootFolder;
00343   std::string sListFilePathName;
00344   std::string sReportRootFolder;
00345   std::string sSubfolder;   // relatively to root list file
00346   std::string sIdealRootFolder;
00347
00348   bool bStopped, bBreak;
00349
00350   std::vector<VarSetT> variables;
00351
00352   GenUniqueInt *pClipNumGenerator;
00353   bool bOwnClipNumGenerator;
00354
00355   std::ostream *pOsFileListReport;
00356   std::vector<std::string> *pFileListContainer;
00357   bool bDoNotRecog;
00358
00359   // todo: extend - time of processing; first and last negative retcodes etc.
00360 //  int argc;
00361 //  char** argv;
00362   int flags;
00363
00364   proc_list_data(int _flags=0):
00365       flags(_flags), files_ok(0), files_failed(0),
00366       total_files_ok(0), total_files_failed(0),
00367       last_file_seconds(0), list_seconds(0), total_seconds(0),
00368       cur_list_done(0), cur_list_total(0), bStopped(false), bBreak(false), variables(1),
00369       pOsFileListReport(NULL),
00370       pFileListContainer(NULL),
00371       bDoNotRecog(false),
00372       pClipNumGenerator(new GenUniqueInt(1)),
00373       bOwnClipNumGenerator(true)
00374     {}
00375   proc_list_data(
00376     const proc_list_data &parentPLD,
00377     const char *pszImgFolder,
00378     const char *pszLstPathName) :
00379       flags(parentPLD.flags), files_ok(0), files_failed(0),
```

```
00380       total_files_ok(0), total_files_failed(0),
00381       last_file_seconds(0), list_seconds(0), total_seconds(0),
00382       cur_list_done(0), cur_list_total(0), bStopped(false), bBreak(false),
00383       variables(parentPLD.variables),
00384       bDoNotRecog(parentPLD.bDoNotRecog),
00385       pClipNumGenerator(parentPLD.pClipNumGenerator),
00386       bOwnClipNumGenerator(false),
00387       sImgRootFolder(parentPLD.sImgRootFolder + "/" + pszImgFolder),
00388       sReportRootFolder(parentPLD.sReportRootFolder + "/" + pszImgFolder),
00389       sListFilePathName(pszLstPathName), margin(parentPLD.margin + "\t"),
00390       pOsFileListReport(parentPLD.pOsFileListReport),
00391       pFileListContainer(parentPLD.pFileListContainer),
00392       sSubfolder(parentPLD.sSubfolder.empty() ? pszImgFolder : parentPLD.sSubfolder + "/" + pszImgFolder),
00393       sIdealRootFolder(parentPLD.sIdealRootFolder)
00394     {}
00395   ~proc_list_data() {
00396     if (bOwnClipNumGenerator && pClipNumGenerator)
00397       delete pClipNumGenerator;
00398   }
00399 };
00400
00401 class ProcFile
00402 {
00403 public:
00404   class RunParams
00405   {
00406   public:
00407     RunParams(const VarSetT &v)
00408       : pszFilePathName(NULL), pszSymmetricPathFromLst(NULL),
00409       pszLstFolder(NULL), pszLstPathName(NULL), pszRelativePath(NULL), flags(0), vars(v) {}
00410     const char *pszFilePathName;
00411     const char *pszSymmetricPathFromLst;
00412     const char *pszIdealFolder;
00413     const char *pszReportRoot;
00414     const char *pszLstFolder;
00415     const char *pszLstPathName;
00416     const char *pszRelativePath;
00417     int flags;
00418     const VarSetT &vars;
00419   };
00420
00421   int processFile(proc_list_data& pld, const pathname &fileName)
00422   {
00423     using std::string;
00424     string ifile = fileName.is_relative() ? pld.sImgRootFolder + '/' + fileName.path : fileName.path;
00425     pathname pathListFile = pld.sListFilePathName.c_str();
00426     string file_folder = (fileName.folder.empty() ? "" : "/" + fileName.folder);
00427     string sSymmetricPathFromLst = pathListFile.folder + file_folder;
00428     RunParams params(pld.variables.back());
00429     params.pszFilePathName = ifile.c_str();
00430     params.pszSymmetricPathFromLst = sSymmetricPathFromLst.c_str();
00431     params.pszLstFolder = pathListFile.folder.c_str();
00432     params.pszLstPathName = pathListFile.path.c_str();
00433     params.pszReportRoot = pld.sReportRootFolder.c_str();
00434     string sRelativePath = pld.sSubfolder.empty() ? fileName.path : pld.sSubfolder + "/" + fileName.path;
00435     params.pszRelativePath = sRelativePath.c_str();
00436     string sIdealFolder = pld.sIdealRootFolder.empty() ?
00437       sSymmetricPathFromLst
00438       : joinPath(pld.sIdealRootFolder, pld.sSubfolder + file_folder);
00439     params.pszIdealFolder = sIdealFolder.c_str();
00440     params.flags = pld.flags;
00441     long long t_start_item = highResolutionClock();
00442     int res = runAndReport(
00443       params,
00444       pld.pOsFileListReport,
00445       pld.pFileListContainer,
00446       pld.bDoNotRecog);
00447     pld.last_file_seconds = (highResolutionClock()-t_start_item)/double( highResolutionClocksPerSecond() );
00448     pld.list_seconds += pld.last_file_seconds;
00449     pld.total_seconds += pld.last_file_seconds;
00450     if (res < 0) {
00451       pld.files_failed++;
00452       pld.total_files_failed++;
00453     } else {
00454       pld.files_ok++;
00455       pld.total_files_ok++;
00456     }
00457     return res;
00458   }
00459
00460   virtual int run(const RunParams &params)
00461   {
00462     return NO_ERRORS;
00463   }
00464   virtual int finish(const VarSetT &vars, const char *pszLstPath) {
00465     return NO_ERRORS;
00466   }
```

```
00467   int runAndReport(const RunParams &params,
00468     std::ostream *pOs,
00469     std::vector<std::string> *pListFileContainer,
00470     bool bDoNotRecog)
00471   {
00472     if (pOs)
00473       (*pOs) << params.pszFilePathName << std::endl;
00474     if (pListFileContainer)
00475       pListFileContainer->push_back(params.pszFilePathName);
00476     if (!bDoNotRecog)
00477       return run(params);
00478     else
00479       return NO_ERRORS;
00480   }
00481 };
00482
00483 inline
00484 std::string preprocCmdString(const char *pszCmdString) {
00485   std::string sTmp = substrex(pszCmdString, 0, std::string(pszCmdString).find_first_of(';'));
00486   sTmp = substrex(sTmp, 0, sTmp.find_last_not_of("\n\r") + 1);
00487   return substrex(sTmp, 0, sTmp.find_last_not_of(' ') + 1);
00488 }
00489
00490 class ListFileCommand
00491 {
00492 public:
00493   ListFileCommand() {}
00494   virtual ~ListFileCommand() {}
00495   virtual int execute(ProcFile &procFile, proc_list_data& pld) const = 0;
00496   virtual void writeToConsole(proc_list_data& pld) const {}
00497   virtual const char *statement() const =0;
00498   virtual ListFileCommand *construct() const =0;
00499   void preprocAndParseCmdString(const char *pszCmdString) {
00500     parseCmdString(preprocCmdString(pszCmdString).c_str());
00501   }
00502   virtual void parseCmdString(const char *pszCmdString) {
00503     std::string sParams(pszCmdString);
00504     if (statement())
00505       sParams = substrex(sParams, sParams.find_first_not_of(' ', 1 + strlen(statement())));
00506     parseCmdParams(sParams.c_str());
00507   }
00508   ListFileCommand *constructFromString(const char *pszCmdString) const {
00509     ListFileCommand *pCmd = construct();
00510     pCmd->preprocAndParseCmdString(pszCmdString);
00511     return pCmd;
00512   }
00513   virtual void parseCmdParams(const char *pszCmdParams) {}
00514 };
00515
00516 class LFCProcessFile : public ListFileCommand
00517 {
00518 public:
00519   LFCProcessFile() {}
00520   int execute(ProcFile &procFile, proc_list_data& pld) const {
00521     return procFile.processFile(pld, fileName);
00522   }
00523   virtual void writeToConsole(proc_list_data& pld) const {
00524     std::cout << pld.margin << fileName.path << "...\n";
00525   }
00526   LFCProcessFile *construct() const { return new LFCProcessFile; }
00527   const char *statement() const { return NULL; }
00528   void parseCmdParams(const char *pszCmdParams) {
00529     fileName.parse(pszCmdParams);
00530   }
00531 protected:
00532   pathname fileName;
00533 };
00534
00535 class LFCStop : public ListFileCommand
00536 {
00537 public:
00538   int execute(ProcFile &procFile, proc_list_data& pld) const {
00539     pld.bStopped = true;
00540     return NO_ERRORS;
00541   }
00542   virtual void writeToConsole(proc_list_data& pld) const {
00543     std::cout << pld.margin << "Processing stopped\n";
00544   }
00545   ListFileCommand *construct() const { return new LFCStop; }
00546   const char *statement() const { return "stop"; }
00547 };
00548
00549 class LFCBreak : public ListFileCommand
00550 {
00551 public:
00552   LFCBreak() {}
00553   int execute(ProcFile &procFile, proc_list_data& pld) const {
```

```
00554     pld.bBreak = true;
00555     return NO_ERRORS;
00556   }
00557   virtual void writeToConsole(proc_list_data& pld) const {
00558     std::cout << pld.margin << "Breaking lst file\n";
00559   }
00560   ListFileCommand *construct() const { return new LFCBreak; }
00561   const char *statement() const { return "break"; }
00562 };
00563
00564 const char CLIPNUMBER_VAR_NAME[] = "clipNumber";
00565
00566 #if 0
00567 class LFCStartClip : public ListFileCommand
00568 {
00569 public:
00570   LFCStartClip() {}
00571   int execute(ProcFile &procFile, proc_list_data& pld) const {
00572     VarSetT::iterator it = pld.variables.back().find(CLIPNUMBER_VAR_NAME);
00573     if (it != pld.variables.back().end())
00574       it->second = CommonValue(pld.pClipNumGenerator->generate());
00575     else
00576       pld.variables.back()[CLIPNUMBER_VAR_NAME] = pld.pClipNumGenerator->generate();  // zero value of
      clipNumber means there is no clip
00577     return NO_ERRORS;
00578   }
00579   virtual void writeToConsole(proc_list_data& pld) const {
00580     std::cout << pld.margin << "Starting new clip\n";
00581   }
00582   ListFileCommand *construct() const { return new LFCStartClip; }
00583   const char *statement() const { return "start_clip"; }
00584 };
00585
00586 class LFCFinishClip : public ListFileCommand
00587 {
00588 public:
00589   LFCFinishClip() {}
00590   int execute(ProcFile &procFile, proc_list_data& pld) const {
00591     pld.variables.back()[CLIPNUMBER_VAR_NAME] = CommonValue(static_cast<int32_t>(0));
00592     return NO_ERRORS;
00593   }
00594   virtual void writeToConsole(proc_list_data& pld) const {
00595     std::cout << pld.margin << "Clip finished\n";
00596   }
00597   ListFileCommand *construct() const { return new LFCFinishClip; }
00598   const char *statement() const { return "finish_clip"; }
00599 };
00600 #endif
00601
00602 class LFCInclude : public ListFileCommand
00603 {
00604 public:
00605   LFCInclude() {}
00606   int execute(ProcFile &procFile, proc_list_data& pld) const;
00607   virtual void writeToConsole(proc_list_data& pld) const {
00608     std::cout << pld.margin << "Including " << fileName.path << "\n";
00609   }
00610   ListFileCommand *construct() const { return new LFCInclude; }
00611   const char *statement() const { return "include"; }
00612   virtual void parseCmdString(const char *pszCmdString) {
00613     std::string sParams(pszCmdString);
00614     if ((pszCmdString[0] == '#') && statement())
00615       sParams = substrex(sParams, sParams.find_first_not_of(' ', 1 + strlen(statement())));
00616     parseCmdParams(sParams.c_str());
00617   }
00618   void parseCmdParams(const char *pszCmdParams) {
00619     fileName.parse(pszCmdParams);
00620   }
00621 protected:
00622   pathname fileName;
00623 };
00624
00625 class LFCClip : public ListFileCommand
00626 {
00627 public:
00628   LFCClip() {}
00629   int execute(ProcFile &procFile, proc_list_data& pld) const;
00630   virtual void writeToConsole(proc_list_data& pld) const {
00631     std::cout << pld.margin << "Including clip " << fileName.path << "\n";
00632   }
00633   ListFileCommand *construct() const { return new LFCClip; }
00634   const char *statement() const { return "clip"; }
00635   void parseCmdParams(const char *pszCmdParams) {
00636     fileName.parse(pszCmdParams);
00637   }
00638 protected:
00639   pathname fileName;
```

```
00640 };
00641
00642 class LFCEcho : public ListFileCommand
00643 {
00644 public:
00645   LFCEcho() {}
00646   int execute(ProcFile &procFile, proc_list_data& pld) const {
00647     std::cerr << pld.margin << sText << '\n';
00648     return NO_ERRORS;
00649   }
00650   ListFileCommand *construct() const { return new LFCEcho; }
00651   const char *statement() const { return "echo"; }
00652   void parseCmdParams(const char *pszCmdParams) {
00653     sText = pszCmdParams;
00654   }
00655 protected:
00656   std::string sText;
00657 };
00658
00659 class LFCSet : public ListFileCommand
00660 {
00661 public:
00662   LFCSet() {}
00663   int execute(ProcFile &procFile, proc_list_data& pld) const {
00664     for (VarSetT::const_iterator it = variables.begin(); it != variables.end(); it++)
00665       pld.variables.back()[it->first] = it->second;
00666     return NO_ERRORS;
00667   }
00668   ListFileCommand *construct() const { return new LFCSet; }
00669   const char *statement() const { return "set"; }
00670   void parseCmdParams(const char *pszCmdParams) {
00671     using std::string;
00672     string sParams(pszCmdParams);
00673     int startPos = 0;
00674     while(startPos != string::npos) {
00675       int endPos = 0;
00676       string sVarName = substrex(sParams, startPos, (endPos = sParams.find_first_of(" =", startPos)) -
      startPos);
00677       startPos = endPos;
00678       startPos = sParams.find_first_not_of(' ', startPos);
00679       string sVarValue;
00680       startPos++;
00681       if( (startPos != string::npos) && (sParams[startPos] == '"'))
00682         sVarValue = substrex(sParams, startPos, (endPos = sParams.find_first_of('"', startPos + 1) + 1) -
      startPos);
00683       else
00684         sVarValue = substrex(sParams, startPos, (endPos = sParams.find_first_of(" \t", startPos)) -
      startPos);
00685       variables[sVarName] = from_str(sVarValue.c_str());
00686       startPos = endPos;
00687       startPos = sParams.find_first_not_of(" \t", startPos);
00688     }
00689   }
00690 protected:
00691   VarSetT variables;
00692 };
00693
00694 class LFCReset : public ListFileCommand
00695 {
00696 public:
00697   LFCReset() {}
00698   int execute(ProcFile &procFile, proc_list_data& pld) const {
00699     pld.variables.clear();
00700     return NO_ERRORS;
00701   }
00702   ListFileCommand *construct() const { return new LFCReset; }
00703   const char *statement() const { return "reset"; }
00704 };
00705
00706 class LFCPush : public ListFileCommand
00707 {
00708 public:
00709   LFCPush() {}
00710   int execute(ProcFile &procFile, proc_list_data& pld) const {
00711     pld.variables.push_back(VarSetT());
00712     std::vector<VarSetT>::iterator last = pld.variables.end();
00713     last--;
00714     std::vector<VarSetT>::iterator lastButOne = last;
00715     lastButOne--;
00716     *last = *lastButOne;
00717     return NO_ERRORS;
00718   }
00719   ListFileCommand *construct() const { return new LFCPush; }
00720   const char *statement() const { return "push"; }
00721 };
00722
00723 class LFCPop : public ListFileCommand
```

```
00724 {
00725 public:
00726   LFCPop() {}
00727   int execute(ProcFile &procFile, proc_list_data& pld) const {
00728     pld.variables.pop_back();
00729     return NO_ERRORS;
00730   }
00731   ListFileCommand *construct() const { return new LFCPop; }
00732   const char *statement() const { return "pop"; }
00733 };
00734
00735 class listfile // list of terminal filenames of nested listfiles names
00736                // NOTE: filenames may be relative or absolute
00737                // in case of relative filename it's path computed from the listfile location
00738 {
00739 //  std::vector< std::string > filenames;
00740   std::vector<ListFileCommand*> commandList;
00741   typedef std::map<std::string, ListFileCommand*> CommandRegistry;
00742   CommandRegistry commandRegistry;
00743 public:
00744   listfile( const char* listfilename )
00745   {
00746     using std::string;
00747
00748     string line;
00749 //    filenames.clear();
00750     registerCommands();
00751     std::ifstream infile(listfilename, std::ios_base::in);
00752     if (!infile)
00753       std::cout << "Error when open listfile:" << listfilename << std::endl;
00754     while (getline(infile, line, '\n'))
00755     {
00756       string sLine = preprocCmdString(line.c_str());
00757       if (sLine.empty()) // skip empty and commented lines
00758         continue;
00759 //      if (line.[line.length() - 1] == '\r')
00760 //        line.erase(line.length() - 1);
00761
00762       string sCmdName;
00763       if (line[0] == '#')
00764         sCmdName = line.substr(1, line.find_first_of(' ') - 1);
00765       ListFileCommand *pCmd = NULL;
00766       if (sCmdName.empty())
00767       {
00768         pathname filePathName = line.c_str();
00769         if (filePathName.ext == "lst")
00770           pCmd = new LFCInclude;
00771         else
00772           pCmd = new LFCProcessFile;
00773       } else
00774       {
00775         CommandRegistry::iterator itCmd = commandRegistry.find(sCmdName);
00776         if (itCmd != commandRegistry.end())
00777           pCmd = itCmd->second->construct();
00778       }
00779       if (pCmd)
00780       {
00781         pCmd->preprocAndParseCmdString(line.c_str());
00782         commandList.push_back(pCmd);
00783       }
00784 //      if (line == "#stop") // ignore rest of list
00785 //        break;
00786 //      filenames.push_back (line);
00787     }
00788   }
00789   ~listfile() {
00790     using std::vector;
00791     using std::map;
00792     for (vector<ListFileCommand*>::iterator it = commandList.begin(); it != commandList.end(); it++)
00793       if (*it)
00794         delete *it;
00795     for (CommandRegistry::iterator it = commandRegistry.begin(); it != commandRegistry.end(); it++)
00796       if (it->second)
00797         delete it->second;
00798   }
00799   void registerCommands() {
00800     using std::vector;
00801     vector<ListFileCommand*> cmds;
00802     cmds.push_back(new LFCProcessFile);
00803     cmds.push_back(new LFCStop);
00804     cmds.push_back(new LFCBreak);
00805     cmds.push_back(new LFCInclude);
00806     cmds.push_back(new LFCEcho);
00807     cmds.push_back(new LFCSet);
00808     cmds.push_back(new LFCReset);
00809     cmds.push_back(new LFCPush);
00810     cmds.push_back(new LFCPop);
```

```
00811     cmds.push_back(new LFCClip);
00812     for (vector<ListFileCommand*>::iterator it = cmds.begin(); it != cmds.end(); it++)
00813       if ((*it)->statement())
00814         commandRegistry[(*it)->statement()] = *it;
00815       else
00816         commandRegistry[""] = *it;
00817   }
00818   int size() { return commandList.size(); }
00819   const ListFileCommand *operator [] ( int i ) { return commandList[i]; }
00820
00821 }; // class listfile
00822
00823 // proc_list_file flags
00824 const int plf_verbose = 0x001; // if set uses std::cout for output
00825 const int plf_stop_if_result_less_zero = 0x002; // stop processing if returned code less zero
00826
00827
00828 //template <class procfile>  // custom p_procfile->run() called for terminal filenames
00829 inline
00830 int proc_list_file( // recursively parses listfile and nested lists, calls procfile.run()
00831   const char* listfile_or_terminal_file_name, // .lst or other (say .jpg)
00832   proc_list_data& pld,
00833   ProcFile* p_procfile )  // p_procfile->run()
00834 {
00835 //  long long t_start = highResolutionClock();
00836
00837   pathname fn( listfile_or_terminal_file_name ); // "C:\\ququ\\file.ext" => "C:\\ququ" "file.ext"
      ...
00838   pld.sListFilePathName = fn.path;
00839   if (pld.sImgRootFolder.empty())
00840     pld.sImgRootFolder = fn.folder;
00841   if (pld.sReportRootFolder.empty())
00842     pld.sReportRootFolder = pld.sImgRootFolder;
00843
00844   int res=0;
00845   if (fn.ext == "lst")
00846   {
00847     listfile lf( fn.path.c_str() );
00848     if (pld.flags & plf_verbose)
00849     {
00850       std::cout << pld.margin << "========================================\n";
00851       std::cout << pld.margin << "== run list file '" << fn.name << "' of " << lf.size() << " items: \n";
00852       // full name... std::cout << listfile_or_terminal_file_name << "\n";
00853     }
00854
00855 //    pld.margin.push_back('\t');
00856
00857     for (int i=0; i<lf.size(); i++)
00858     {
00859       if (pld.bStopped || pld.bBreak)
00860         break;
00861       pld.cur_list_done = i+1;
00862       pld.cur_list_total = lf.size();
00863 //      pathname fn_i( lf[i].c_str() );
00864       const ListFileCommand *pCmd = lf[i];
00865       if (!pCmd)
00866         continue;
00867 //      std::string ifile = fn_i.is_relative() ? fn.folder + '/' + lf[i] : lf[i];
00868       if (pld.flags & plf_verbose)
00869       {
00870         std::cout << pld.margin << "{{{ list item " << pld.cur_list_done << " of " << pld.cur_list_total <<
      "\n";
00871         pCmd->writeToConsole(pld);
00872 //        std::cout << pld.margin << ((fn_i.ext == "lst") ? ifile : lf[i]) << "...\n";
00873       }
00874 //      long long t_start_item = highResolutionClock();
00875       if (pCmd->execute(*p_procfile, pld) < 0)
00876         std::cout << "Command execution failed\n";
00877 //      res = proc_list_file<procfile>( ifile.c_str(), pld, p_procfile );
00878
00879
00880       if (pld.flags & plf_verbose)
00881       {
00882 //        double sec = (highResolutionClock()-t_start_item)/double( highResolutionClocksPerSecond() );
00883         double ave = pld.files_ok+pld.files_failed > 0 ? pld.list_seconds / (pld.files_ok+pld.files_failed)
      : pld.list_seconds;
00884         std::cout << pld.margin << "...finished in "<< pld.last_file_seconds  <<" sec; retcode=" << res <<
     "\n";
00885         std::cout << pld.margin << "subtotal: files_ok=" << pld.files_ok << " files_failed=" << pld.
     files_failed << "\n";
00886         std::cout << pld.margin << "average time: " << ave << " sec\n";
00887         std::cout << pld.margin << "}}}\n";
00888       }
00889
00890       if (res < 0 && ( pld.flags & plf_stop_if_result_less_zero))
00891         break;
00892     } //for (int i=0; i<lf.size(); i++)
```

```
00893     p_procfile->finish(pld.variables.back(), fn.folder.c_str());
00894
00895     if (pld.margin.length() > 0) // pop_back()
00896       pld.margin.resize( pld.margin.length()-1 );
00897       //pld.margin=pld.margin.substr( 0, pld.margin.length()-1 );
00898
00899     if (pld.flags & plf_verbose)
00900     {
00901   //    double sec = (highResolutionClock()-t_start)/double( highResolutionClocksPerSecond() );
00902       double ave = pld.files_ok+pld.files_failed > 0 ? pld.list_seconds / (pld.files_ok+pld.files_failed) :
      pld.list_seconds;
00903       std::cout << pld.margin << "== end of list file '" << fn.name << "'\n";
00904       std::cout << pld.margin << "== files_ok=" << pld.files_ok << " files_failed=" << pld.files_failed <<
     "\n";
00905       std::cout << pld.margin << "== average time=" << ave << " sec\n";
00906       std::cout << pld.margin << "==========================================\n";
00907     }
00908   }
00909   else // not nested .lst file - let process item
00910   {
00911
00912     return INTERNAL_ERROR;
00913  }
00914
00915   return res;
00916 }
00917
00918 inline
00919 int LFCInclude::execute(ProcFile &procFile, proc_list_data& pld) const {
00920   pathname pathListFile = pld.sListFilePathName.c_str();
00921   pathname ifile = (fileName.is_relative() ? pathListFile.folder + '/' + fileName.path : fileName.
     path).c_str();
00922   proc_list_data newPld(pld, fileName.folder.c_str(), ifile.path.c_str());
00923 /*  newPld.variables = pld.variables;
00924   newPld.sImgRootFolder = pld.sImgRootFolder + "/" + fileName.folder;
00925   newPld.sListFileFolder = ifile.folder;
00926   newPld.margin = pld.margin + "\t";
00927   newPld.bDoNotRecog = pld.bDoNotRecog;
00928   newPld.pOsFileListReport = pld.pOsFileListReport;*/
00929   int res = proc_list_file(ifile.path.c_str(), newPld, &procFile);
00930   if (newPld.bStopped)
00931     pld.bStopped = true;
00932   pld.variables = newPld.variables;
00933   pld.total_files_ok += newPld.total_files_ok;
00934   pld.total_files_failed += newPld.total_files_failed;
00935   pld.total_seconds += newPld.total_seconds;
00936   return res;
00937 }
00938
00939 inline
00940 int LFCClip::execute(ProcFile &procFile, proc_list_data& pld) const {
00941   if (fileName.ext == "lst")
00942   {
00943     pathname pathListFile = pld.sListFilePathName.c_str();
00944     pathname ifile((fileName.is_relative() ? pathListFile.folder + '/' + fileName.path : fileName.
     path).c_str());
00945     CommonValue &clipNumber = pld.variables.back()[CLIPNUMBER_VAR_NAME];
00946     bool bClipStarted = true;
00947     if (!clipNumber.empty()
00948       && (clipNumber != CommonValue(static_cast<int32_t>(0))))
00949     {
00950       std::cerr << "Warning: nested #clip - iterpreting as #include\n";
00951       bClipStarted = false;
00952     } else
00953       clipNumber = pld.pClipNumGenerator->generate();
00954     proc_list_data newPld(pld, fileName.folder.c_str(), ifile.path.c_str());
00955     int res = proc_list_file(ifile.path.c_str(), newPld, &procFile);
00956     if (newPld.bStopped)
00957       pld.bStopped = true;
00958     pld.variables = newPld.variables;
00959     pld.total_files_ok += newPld.total_files_ok;
00960     pld.total_files_failed += newPld.total_files_failed;
00961     pld.total_seconds += newPld.total_seconds;
00962     if (bClipStarted)
00963       pld.variables.back()[CLIPNUMBER_VAR_NAME] = CommonValue(static_cast<int32_t>(0));
00964     return res;
00965   } else // not a list considered as picture
00966   {
00967     return procFile.processFile(pld, fileName);
00968   }
00969   return INTERNAL_ERROR;
00970 }
00971
00972 }; // namespace se::
00973
00974 #endif // #ifndef MINUTILS_LISTFILE_H_INCLUDED
```

5.3 mathoper.h File Reference

Definition of mathematical operations.

Enumerations

- enum MathOp

  Specifies mathematical operations.
- enum UnOp

  Specifies unary operations.
- enum BiOp

  Specifies binary operations.
- enum AsOp

  Specifies associative operations.
- enum CoOp

  Specifies commutative operations.
- enum AsCoOp

  Specifies associative-commutative operations.
- enum IdOp

  Specifies idempotent operations.

Functions

- **DECLARE_COMPOUND_WITH_TYPEDEF** (enum, MathOp)
- **DECLARE_COMPOUND_WITH_TYPEDEF** (enum, UnOp)
- **DECLARE_COMPOUND_WITH_TYPEDEF** (enum, BiOp)
- **DECLARE_COMPOUND_WITH_TYPEDEF** (enum, AsOp)
- **DECLARE_COMPOUND_WITH_TYPEDEF** (enum, CoOp)
- **DECLARE_COMPOUND_WITH_TYPEDEF** (enum, AsCoOp)
- **DECLARE_COMPOUND_WITH_TYPEDEF** (enum, IdOp)

5.4 mathoper.h

```
00001 /*
00002
00003 Copyright (c) 2011, Smart Engines Limited. All rights reserved.
00004
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without modification,
00008 are permitted provided that the following conditions are met:
00009
00010    1. Redistributions of source code must retain the above copyright notice,
00011       this list of conditions and the following disclaimer.
00012
00013    2. Redistributions in binary form must reproduce the above copyright notice,
00014       this list of conditions and the following disclaimer in the documentation
00015       and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00020 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027
```

```
00028 The views and conclusions contained in the software and documentation are those
00029 of the authors and should not be interpreted as representing official policies,
00030 either expressed or implied, of copyright holders.
00031
00032 */
00033
00038 #pragma once
00039 #ifndef MINUTILS_MATHOPER_H_INCLUDED
00040 #define MINUTILS_MATHOPER_H_INCLUDED
00041
00057 #include <minbase/macro_helpers.h>
00058
00064 enum MathOp {
00070   OP_MIN = 1,
00071
00077   OP_MAX,
00078
00084   OP_ADD,
00085
00091   OP_DIF,
00092
00098   OP_ADF,
00099
00105   OP_MUL,
00106
00112   OP_AVE,
00113
00119   OP_EUC,
00120
00126   OP_DIV,
00127
00133   OP_SSQ,
00134
00141   OP_ABS,
00142
00148   OP_SQRT,
00149
00155   OP_POW,
00156
00162   OP_INV,
00163
00169   OP_AND,
00170
00176   OP_OR,
00177
00183   OP_XOR,
00184 };
00185
00194 enum UnOp {
00195   UNOP_ABS  = OP_ABS,
00196   UNOP_SQRT = OP_SQRT,
00197   UNOP_INV  = OP_INV,
00198 };
00199
00209 enum BiOp {
00210   BIOP_MIN = OP_MIN,
00211   BIOP_MAX = OP_MAX,
00212   BIOP_ADD = OP_ADD,
00213   BIOP_DIF = OP_DIF,
00214   BIOP_ADF = OP_ADF,
00215   BIOP_MUL = OP_MUL,
00216   BIOP_AVE = OP_AVE,
00217   BIOP_EUC = OP_EUC,
00218   BIOP_DIV = OP_DIV,
00219   BIOP_SSQ = OP_SSQ,
00220   BIOP_POW = OP_POW,
00221   BIOP_AND = OP_AND,
00222   BIOP_OR  = OP_OR,
00223   BIOP_XOR = OP_XOR,
00224 };
00225
00235 enum AsOp {
00236   ASOP_MIN = OP_MIN,
00237   ASOP_MAX = OP_MAX,
00238   ASOP_ADD = OP_ADD,
00239   ASOP_MUL = OP_MUL,
00240   ASOP_EUC = OP_EUC,
00241   ASOP_AND = OP_AND,
00242   ASOP_OR  = OP_OR,
00243   ASOP_XOR = OP_XOR,
00244 };
00245
00255 enum CoOp {
00256   COOP_MIN = OP_MIN,
00257   COOP_MAX = OP_MAX,
00258   COOP_ADD = OP_ADD,
00259   COOP_ADF = OP_ADF,
```

```
00260   COOP_MUL = OP_MUL,
00261   COOP_AVE = OP_AVE,
00262   COOP_EUC = OP_EUC,
00263   COOP_SSQ = OP_SSQ,
00264   COOP_AND = OP_AND,
00265   COOP_OR  = OP_OR,
00266   COOP_XOR = OP_XOR,
00267 };
00268
00280 enum AsCoOp {
00281   ASCOOP_MIN = OP_MIN,
00282   ASCOOP_MAX = OP_MAX,
00283   ASCOOP_ADD = OP_ADD,
00284   ASCOOP_MUL = OP_MUL,
00285   ASCOOP_EUC = OP_EUC,
00286   ASCOOP_AND = OP_AND,
00287   ASCOOP_OR  = OP_OR,
00288   ASCOOP_XOR = OP_XOR,
00289 };
00290
00299 enum IdOp {
00300   IDOP_MIN = OP_MIN,
00301   IDOP_MAX = OP_MAX,
00302   IDOP_AND = OP_AND,
00303   IDOP_OR  = OP_OR,
00304 };
00305
00306 DECLARE_COMPOUND_WITH_TYPEDEF(enum, MathOp);
00307 DECLARE_COMPOUND_WITH_TYPEDEF(enum, UnOp);
00308 DECLARE_COMPOUND_WITH_TYPEDEF(enum, BiOp);
00309 DECLARE_COMPOUND_WITH_TYPEDEF(enum, AsOp);
00310 DECLARE_COMPOUND_WITH_TYPEDEF(enum, CoOp);
00311 DECLARE_COMPOUND_WITH_TYPEDEF(enum, AsCoOp);
00312 DECLARE_COMPOUND_WITH_TYPEDEF(enum, IdOp);
00313
00314 #endif // #ifndef MINUTILS_MATHOPER_H_INCLUDED
```

5.5 minarr.h File Reference

Definition of a multi-dimensional dense multi-channel array.

Data Structures

- struct MinArr

  A multi-dimensional dense multi-channel array representation. More...

5.6 minarr.h

```
00001 /*
00002
00003 Copyright (c) 2011, Smart Engines Limited. All rights reserved.
00004
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without modification,
00008 are permitted provided that the following conditions are met:
00009
00010    1. Redistributions of source code must retain the above copyright notice,
00011       this list of conditions and the following disclaimer.
00012
00013    2. Redistributions in binary form must reproduce the above copyright notice,
00014       this list of conditions and the following disclaimer in the documentation
00015       and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00020 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
```

```
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027
00028 The views and conclusions contained in the software and documentation are those
00029 of the authors and should not be interpreted as representing official policies,
00030 either expressed or implied, of copyright holders.
00031
00032 */
00033
00039 #pragma once
00040 #ifndef MINUTILS_MINARR_H_INCLUDED
00041 #define MINUTILS_MINARR_H_INCLUDED
00042
00043 #include <minbase/mintyp.h>
00044
00069 typedef struct
00070 {
00071   int32_t  dim;
00072   int32_t *pSizes;
00074   int32_t *pStrides;
00076   int32_t  channelDepth;
00078   MinFmt   format;
00079   uint8_t *pStart;
00081 } MinArr;
00082
00083 #endif // #ifndef MINUTILS_MINARR_H_INCLUDED
```

5.7 pathname.h File Reference

Crossplatform parsing filenames to path, name, extension.

Data Structures

- class se::pathname

5.8 pathname.h

```
00001 /*
00002
00003 Copyright (c) 2011, Smart Engines Limited. All rights reserved.
00004
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without modification,
00008 are permitted provided that the following conditions are met:
00009
00010    1. Redistributions of source code must retain the above copyright notice,
00011       this list of conditions and the following disclaimer.
00012
00013    2. Redistributions in binary form must reproduce the above copyright notice,
00014       this list of conditions and the following disclaimer in the documentation
00015       and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00020 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027
00028 The views and conclusions contained in the software and documentation are those
00029 of the authors and should not be interpreted as representing official policies,
00030 either expressed or implied, of copyright holders.
00031
00032 */
00033
00039 #pragma once
00040 #ifndef MINUTILS_PATHNAME_H_INCLUDED
00041 #define MINUTILS_PATHNAME_H_INCLUDED
00042
00043 #include <string>
```

```
00044
00045 namespace se {
00046 class pathname
00047 {
00048 public:
00049   std::string path; // given path
00050   // contains path string that should be parsed
00051   // path = "C:\\ququ\\file.ext" or "..\\ququ\\file.ext"  or "file.ext" etc.
00052
00053   // below results of parsing
00054   std::string folder;       // respectively, results are "C:\\ququ", "..\\ququ", ""
00055   std::string name;         // "file.ext"
00056   std::string name_base;     // "file"
00057   std::string ext;         // "ext"
00058   pathname( const char* str = NULL ) { parse(str); }
00059   void parse( const char* str = NULL )
00060   {
00061     if (str!=NULL)
00062     {
00063       path = str;
00064
00065       size_t found = path.find_last_of("/\\");
00066       if (found == std::string::npos) // slash not found
00067       {
00068         folder = "";
00069         name = path;
00070       }
00071       else
00072       {
00073         folder = path.substr(0,found);
00074         name = path.substr(found+1);
00075       }
00076
00077       size_t found2 = name.find_last_of('.');
00078       name_base = name.substr(0,found2);
00079       ext = name.substr(found2+1);
00080     }
00081   }
00082
00083   bool is_relative() const
00084   {
00085     if (path.empty())
00086       return false;
00087     size_t len = path.length();
00088     if (path[0] == '\\' || path[0] == '/')
00089       return false;
00090     if (len > 2 && path[1] == ':') // c:\sdf\sdf
00091       return false;
00092     // todo: correct for macOS, say "::" and other cases ...
00093     // for different cases may refer to
00094     // http://en.wikipedia.org/wiki/Path_(computing)
00095     return true;
00096   }
00097 }; // class pathname
00098 }; // namespace se::
00099
00100 #endif // #ifndef MINUTILS_PATHNAME_H_INCLUDED
```

5.9 smartptr.h File Reference

Different cross-platform declarations.

Macros

- #define **MINUTILS_SMARTPTR_H_INCLUDED**
- #define DEFINE_SCOPED_OBJECT(name, freeing_proc)

  Defines a template class which will take care about freeing memory of allocated array when the scope of such array ends.
- #define **DEFINE_SCOPED_HANDLE**(name, type, freeing_proc)

Functions

- template<typename TData >
  static MUSTINLINE TData * **ShiftPtr** (TData *ptr, int shift)

5.9.1 Macro Definition Documentation

5.9.1.1 DEFINE_SCOPED_HANDLE

```
#define DEFINE_SCOPED_HANDLE(
              name,
              type,
              freeing_proc )
```

Value:

```
class name {                                          \
   public:                                             \
    name() : handle(NULL) { }                          \
    ~name() { freeing_proc(&handle); }                 \
    operator type () const  { return handle; }         \
    type * get() { return &handle; }                   \
    void free() { freeing_proc(&handle); handle = NULL; } \
    void reset(type new_handle) {                      \
      freeing_proc(&handle);                           \
      handle = new_handle;                             \
    }                                                  \
    type release() {                                   \
      type res = handle;                               \
      handle = NULL;                                   \
      return res;                                      \
    }                                                  \
   private:                                            \
    name(const name &);                                \
    void operator =(const name &);                     \
    type handle;                                       \
  };
```

Definition at line 114 of file smartptr.h.

5.10 smartptr.h

```
00001 /*
00002
00003 Copyright (c) 2011, Smart Engines Limited. All rights reserved.
00004
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without modification, are
00008 permitted provided that the following conditions are met:
00009
00010    1. Redistributions of source code must retain the above copyright notice, this list of
00011       conditions and the following disclaimer.
00012
00013    2. Redistributions in binary form must reproduce the above copyright notice, this list
00014       of conditions and the following disclaimer in the documentation and/or other materials
00015       provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00018 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
00019 FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDERS OR
00020 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00021 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
00022 SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00023 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00024 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00025 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026
00027 The views and conclusions contained in the software and documentation are those of the
00028 authors and should not be interpreted as representing official policies, either expressed
00029 or implied, of copyright holders.
00030
00031 */
00032
00038 #pragma once
00039 #ifndef MINUTILS_SMARTPTR_H_INCLUDED
00040 #define MINUTILS_SMARTPTR_H_INCLUDED
00041
```

```
00042 #include <cstdlib>
00043 #include <minbase/crossplat.h>
00044 #include <minbase/mintyp.h>
00045
00062 #define DEFINE_SCOPED_OBJECT(name, freeing_proc) \
00063 template<typename T> class name \
00064 { \
00065 public: \
00066   explicit name(T *p = 0): p(p) {} \
00067   ~name() \
00068   { \
00069     freeing_proc(p); \
00070   } \
00071   operator T *() const \
00072   { \
00073     return p; \
00074   } \
00075 private: \
00076   name(const name &); \
00077   void operator =(const name &); \
00078   T *p; \
00079 };
00080
00089 DEFINE_SCOPED_OBJECT(scoped_c_array, free)
00090
00091
00099 DEFINE_SCOPED_OBJECT(scoped_cpp_array, delete[])
00100
00101 template<typename TData> static MUSTINLINE TData *ShiftPtr
00102 (
00103   TData *ptr,
00104   int    shift
00105 )
00106 {
00107   return const_cast<TData *>(
00108            reinterpret_cast<const TData *>(
00109              reinterpret_cast<const uint8_t *>(ptr) + shift));
00110 }
00111
00112
00113 // Helper to create scoped handles.
00114 #define DEFINE_SCOPED_HANDLE(name, type, freeing_proc)     \
00115   class name {                                             \
00116    public:                                                 \
00117     name() : handle(NULL) { }                              \
00118     ~name() { freeing_proc(&handle); }                     \
00119     operator type () const  { return handle; }             \
00120     type * get() { return &handle; }                       \
00121     void free() { freeing_proc(&handle); handle = NULL; } \
00122     void reset(type new_handle) {                          \
00123       freeing_proc(&handle);                               \
00124       handle = new_handle;                                 \
00125     }                                                      \
00126     type release() {                                       \
00127       type res = handle;                                   \
00128       handle = NULL;                                       \
00129       return res;                                          \
00130     }                                                      \
00131    private:                                                \
00132     name(const name &);                                    \
00133     void operator =(const name &);                         \
00134     type handle;                                           \
00135   };
00136
00137 #endif // #ifndef MINUTILS_SMARTPTR_H_INCLUDED
```

5.11 minutils.cpp File Reference

Main source file.

5.12 minutils.cpp

```
00001 /*
00002
00003 Copyright (c) 2011, Smart Engines Limited. All rights reserved.
00004
00005 All rights reserved.
00006
```

```
00007 Redistribution and use in source and binary forms, with or without modification,
00008 are permitted provided that the following conditions are met:
00009
00010    1. Redistributions of source code must retain the above copyright notice,
00011       this list of conditions and the following disclaimer.
00012
00013    2. Redistributions in binary form must reproduce the above copyright notice,
00014       this list of conditions and the following disclaimer in the documentation
00015       and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00020 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027
00028 The views and conclusions contained in the software and documentation are those
00029 of the authors and should not be interpreted as representing official policies,
00030 either expressed or implied, of copyright holders.
00031
00032 */
00033
```

5.13 stdint-doc.h File Reference

Documentation for <stdint.h> members.

5.14 stdint-doc.h

```
00001 /*
00002
00003 Copyright (c) 2011, Smart Engines Limited. All rights reserved.
00004
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without modification,
00008 are permitted provided that the following conditions are met:
00009
00010    1. Redistributions of source code must retain the above copyright notice,
00011       this list of conditions and the following disclaimer.
00012
00013    2. Redistributions in binary form must reproduce the above copyright notice,
00014       this list of conditions and the following disclaimer in the documentation
00015       and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00020 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027
00028 The views and conclusions contained in the software and documentation are those
00029 of the authors and should not be interpreted as representing official policies,
00030 either expressed or implied, of copyright holders.
00031
00032 */
00033
00039 #pragma once
00040 #ifndef MINUTILS_STDINT_VC_DOC_H_INCLUDED
00041 #define MINUTILS_STDINT_VC_DOC_H_INCLUDED
00042
00557 #endif // #ifndef MINUTILS_STDINT_VC_DOC_H_INCLUDED
```

5.15 stdint-vc.h File Reference

C99 standard library header file for use with MS VC++.

Macros

- #define _W64

  Defines `_W64` macros to mark types changing their size, like `intptr_t` or `uintptr_t`.
- #define INT8_MIN ((int8_t)_I8_MIN)

  Defines a minimum value of a signed 8-bit integer.
- #define INT8_MAX _I8_MAX

  Defines a maximum value of a signed 8-bit integer.
- #define INT16_MIN ((int16_t)_I16_MIN)

  Defines a minimum value of a signed 16-bit integer.
- #define INT16_MAX _I16_MAX

  Defines a maximum value of a signed 16-bit integer.
- #define INT32_MIN ((int32_t)_I32_MIN)

  Defines a minimum value of a signed 32-bit integer.
- #define INT32_MAX _I32_MAX

  Defines a maximum value of a signed 32-bit integer.
- #define INT64_MIN ((int64_t)_I64_MIN)

  Defines a minimum value of a signed 64-bit integer.
- #define INT64_MAX _I64_MAX

  Defines a maximum value of a signed 64-bit integer.
- #define UINT8_MAX _UI8_MAX

  Defines a maximum value of an unsigned 8-bit integer.
- #define UINT16_MAX _UI16_MAX

  Defines a maximum value of an unsigned 16-bit integer.
- #define UINT32_MAX _UI32_MAX

  Defines a maximum value of an unsigned 32-bit integer.
- #define UINT64_MAX _UI64_MAX

  Defines a maximum value of an unsigned 64-bit integer.
- #define INT_LEAST8_MIN INT8_MIN

  Defines a minimum value of a signed integer with a width of at least 8 bits.
- #define INT_LEAST8_MAX INT8_MAX

  Defines a maximum value of a signed integer with a width of at least 8 bits.
- #define INT_LEAST16_MIN INT16_MIN

  Defines a minimum value of a signed integer with a width of at least 16 bits.
- #define INT_LEAST16_MAX INT16_MAX

  Defines a maximum value of a signed integer with a width of at least 16 bits.
- #define INT_LEAST32_MIN INT32_MIN

  Defines a minimum value of a signed integer with a width of at least 32 bits.
- #define INT_LEAST32_MAX INT32_MAX

  Defines a maximum value of a signed integer with a width of at least 32 bits.
- #define INT_LEAST64_MIN INT64_MIN

  Defines a minimum value of a signed integer with a width of at least 64 bits.
- #define INT_LEAST64_MAX INT64_MAX

  Defines a maximum value of a signed integer with a width of at least 64 bits.
- #define UINT_LEAST8_MAX UINT8_MAX

  Defines a maximum value of an unsigned integer with a width of at least 8 bits.
- #define UINT_LEAST16_MAX UINT16_MAX

  Defines a maximum value of an unsigned integer with a width of at least 16 bits.
- #define UINT_LEAST32_MAX UINT32_MAX

  Defines a maximum value of an unsigned integer with a width of at least 32 bits.
- #define UINT_LEAST64_MAX UINT64_MAX

Defines a maximum value of an unsigned integer with a width of at least 64 bits.

- #define INT_FAST8_MIN INT8_MIN

  Defines a minimum value of a fastest signed 8-bit integer.
- #define INT_FAST8_MAX INT8_MAX

  Defines a maximum value of a fastest signed 8-bit integer.
- #define INT_FAST16_MIN INT16_MIN

  Defines a minimum value of a fastest signed 16-bit integer.
- #define INT_FAST16_MAX INT16_MAX

  Defines a maximum value of a fastest signed 16-bit integer.
- #define INT_FAST32_MIN INT32_MIN

  Defines a minimum value of a fastest signed 32-bit integer.
- #define INT_FAST32_MAX INT32_MAX

  Defines a maximum value of a fastest signed 32-bit integer.
- #define INT_FAST64_MIN INT64_MIN

  Defines a minimum value of a fastest signed 64-bit integer.
- #define INT_FAST64_MAX INT64_MAX

  Defines a maximum value of a fastest signed 64-bit integer.
- #define UINT_FAST8_MAX UINT8_MAX

  Defines a maximum value of a fastest unsigned 8-bit integer.
- #define UINT_FAST16_MAX UINT16_MAX

  Defines a maximum value of a fastest unsigned 16-bit integer.
- #define UINT_FAST32_MAX UINT32_MAX

  Defines a maximum value of a fastest unsigned 32-bit integer.
- #define UINT_FAST64_MAX UINT64_MAX

  Defines a maximum value of a fastest unsigned 64-bit integer.
- #define INTPTR_MIN INT32_MIN

  Defines a minimum value of a signed integer which is guaranteed to hold the value of a pointer.
- #define INTPTR_MAX INT32_MAX

  Defines a maximum value of a signed integer which is guaranteed to hold the value of a pointer.
- #define UINTPTR_MAX UINT32_MAX

  Defines a maximum value of an unsigned integer which is guaranteed to hold the value of a pointer.
- #define INTMAX_MIN INT64_MIN

  Defines a minimum value of a signed integer which has the greatest limits.
- #define INTMAX_MAX INT64_MAX

  Defines a maximum value of a signed integer which has the greatest limits.
- #define UINTMAX_MAX UINT64_MAX

  Defines a maximum value of an unsigned integer which has the greatest limits.
- #define PTRDIFF_MIN _I32_MIN

  Defines a minimum value `ptrdiff_t` can hold.
- #define PTRDIFF_MAX _I32_MAX

  Defines a maximum value `ptrdiff_t` can hold.
- #define SIG_ATOMIC_MIN INT_MIN

  Defines a minimum value `sig_atomic_t` can hold.
- #define SIG_ATOMIC_MAX INT_MAX

  Defines a maximum value `sig_atomic_t` can hold.
- #define SIZE_MAX _UI32_MAX

  Defines a maximum value `size_t` can hold.
- #define WCHAR_MIN 0

  Defines a minimum value for type `wchar_t`.
- #define WCHAR_MAX _UI16_MAX

  Defines a maximum value for type `wchar_t`.

- #define WINT_MIN 0

  Defines a minimum value for type `wint_t`.
- #define WINT_MAX _UI16_MAX

  Defines a maximum value for type `wint_t`.
- #define INT8_C(val) val##i8

  Defines a macros which converts an integer literal to a signed integer with a width of at least 8 bits.
- #define INT16_C(val) val##i16

  Defines a macros which converts an integer literal to a signed integer with a width of at least 16 bits.
- #define INT32_C(val) val##i32

  Defines a macros which converts an integer literal to a signed integer with a width of at least 32 bits.
- #define INT64_C(val) val##i64

  Defines a macros which converts an integer literal to a signed integer with a width of at least 64 bits.
- #define UINT8_C(val) val##ui8

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 8 bits.
- #define UINT16_C(val) val##ui16

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 16 bits.
- #define UINT32_C(val) val##ui32

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 32 bits.
- #define UINT64_C(val) val##ui64

  Defines a macros which converts an integer literal to an unsigned integer with a width of at least 64 bits.
- #define INTMAX_C INT64_C

  Defines a macros which converts an integer literal to a signed integer which has the greatest limits.
- #define UINTMAX_C UINT64_C

  Defines a macros which converts an integer literal to an unsigned integer which has the greatest limits.

Typedefs

- typedef signed char int8_t

  Defines a signed integer type with a width of exactly 8 bits.
- typedef signed short int16_t

  Defines a signed integer type with a width of exactly 16 bits.
- typedef signed int int32_t

  Defines a signed integer type with a width of exactly 32 bits.
- typedef unsigned char uint8_t

  Defines an unsigned integer type with a width of exactly 8 bits.
- typedef unsigned short uint16_t

  Defines an unsigned integer type with a width of exactly 16 bits.
- typedef unsigned int uint32_t

  Defines an unsigned integer type with a width of exactly 32 bits.
- typedef signed __int64 int64_t

  Defines a signed integer type with a width of exactly 64 bits.
- typedef unsigned __int64 uint64_t

  Defines an unsigned integer type with a width of exactly 64 bits.
- typedef int8_t int_least8_t

  Defines a signed integer type with a width of at least 8 bits.
- typedef int16_t int_least16_t

  Defines a signed integer type with a width of at least 16 bits.
- typedef int32_t int_least32_t

  Defines a signed integer type with a width of at least 32 bits.
- typedef int64_t int_least64_t

Defines a signed integer type with a width of at least 64 bits.

- typedef uint8_t uint_least8_t

  Defines an unsigned integer type with a width of at least 8 bits.
- typedef uint16_t uint_least16_t

  Defines an unsigned integer type with a width of at least 16 bits.
- typedef uint32_t uint_least32_t

  Defines an unsigned integer type with a width of at least 32 bits.
- typedef uint64_t uint_least64_t

  Defines an unsigned integer type with a width of at least 64 bits.
- typedef int8_t int_fast8_t

  Defines a signed integer type being usually fastest with a width of at least 8 bits.
- typedef int16_t int_fast16_t

  Defines a signed integer type being usually fastest with a width of at least 16 bits.
- typedef int32_t int_fast32_t

  Defines a signed integer type being usually fastest with a width of at least 32 bits.
- typedef int64_t int_fast64_t

  Defines a signed integer type being usually fastest with a width of at least 64 bits.
- typedef uint8_t uint_fast8_t

  Defines an unsigned integer type being usually fastest with a width of at least 8 bits.
- typedef uint16_t uint_fast16_t

  Defines an unsigned integer type being usually fastest with a width of at least 16 bits.
- typedef uint32_t uint_fast32_t

  Defines an unsigned integer type being usually fastest with a width of at least 32 bits.
- typedef uint64_t uint_fast64_t

  Defines an unsigned integer type being usually fastest with a width of at least 64 bits.
- typedef _W64 signed int intptr_t

  Defines a signed integer type which is guaranteed to hold the value of a pointer.
- typedef _W64 unsigned int uintptr_t

  Defines an unsigned integer type which is guaranteed to hold the value of a pointer.
- typedef int64_t intmax_t

  Defines a signed integer type which has the greatest limits.
- typedef uint64_t uintmax_t

  Defines an unsigned integer type which has the greatest limits.

5.15.1 Detailed Description

Author

Alexander Chemeris

Definition in file stdint-vc.h.

5.16 stdint-vc.h

```
00001 // ISO C9x  compliant stdint.h for Microsoft Visual Studio
00002 // Based on ISO/IEC 9899:TC2 Committee draft (May 6, 2005) WG14/N1124
00003 //
00004 //  Copyright (c) 2006-2008 Alexander Chemeris
00005 //
00006 // Redistribution and use in source and binary forms, with or without
00007 // modification, are permitted provided that the following conditions are met:
00008 //
00009 //   1. Redistributions of source code must retain the above copyright notice,
00010 //      this list of conditions and the following disclaimer.
00011 //
00012 //   2. Redistributions in binary form must reproduce the above copyright
00013 //      notice, this list of conditions and the following disclaimer in the
00014 //      documentation and/or other materials provided with the distribution.
00015 //
00016 //   3. The name of the author may be used to endorse or promote products
00017 //      derived from this software without specific prior written permission.
00018 //
00019 // THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 // WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 // MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00022 // EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 // PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
00025 // OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
00026 // WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
00027 // OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00028 // ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00029 //
00031
00032 #ifndef _MSC_VER // [
00033 #error "Use this header only with Microsoft Visual C++ compilers!"
00034 #endif // _MSC_VER ]
00035
00036 #ifndef _MSC_STDINT_H_ // [
00037 #define _MSC_STDINT_H_
00038
00039 #if _MSC_VER > 1000
00040 #pragma once
00041 #endif
00042
00043 #include <limits.h>
00044
00045 // For Visual Studio 6 in C++ mode and for many Visual Studio versions when
00046 // compiling for ARM we should wrap <wchar.h> include with 'extern "C++" {}'
00047 // or compiler give many errors like this:
00048 //   error C2733: second C linkage of overloaded function 'wmemchr' not allowed
00049 #ifdef __cplusplus
00050 extern "C++" {
00051 #endif
00052 #  include <wchar.h>
00053 #ifdef __cplusplus
00054 }
00055 #endif
00056
00057 // Define _W64 macros to mark types changing their size, like intptr_t.
00058 #ifndef _W64
00059 #  if !defined(__midl) && (defined(_X86_) || defined(_M_IX86)) && _MSC_VER >= 1300
00060 #     define _W64 __w64
00061 #  else
00062 #     define _W64
00063 #  endif
00064 #endif
00065
00066
00067 // 7.18.1 Integer types
00068
00069 // 7.18.1.1 Exact-width integer types
00070
00071 // Visual Studio 6 and Embedded Visual C++ 4 doesn't
00072 // realize that, e.g. char has the same size as __int8
00073 // so we give up on __intX for them.
00074 #if (_MSC_VER < 1300)
00075    typedef signed char       int8_t;
00076    typedef signed short      int16_t;
00077    typedef signed int        int32_t;
00078    typedef unsigned char     uint8_t;
00079    typedef unsigned short    uint16_t;
00080    typedef unsigned int      uint32_t;
00081 #else
00082    typedef signed __int8     int8_t;
00083    typedef signed __int16    int16_t;
00084    typedef signed __int32    int32_t;
00085    typedef unsigned __int8   uint8_t;
```

```
00086    typedef unsigned __int16  uint16_t;
00087    typedef unsigned __int32  uint32_t;
00088 #endif
00089 typedef signed __int64       int64_t;
00090 typedef unsigned __int64     uint64_t;
00091
00092
00093 // 7.18.1.2 Minimum-width integer types
00094 typedef int8_t    int_least8_t;
00095 typedef int16_t   int_least16_t;
00096 typedef int32_t   int_least32_t;
00097 typedef int64_t   int_least64_t;
00098 typedef uint8_t   uint_least8_t;
00099 typedef uint16_t  uint_least16_t;
00100 typedef uint32_t  uint_least32_t;
00101 typedef uint64_t  uint_least64_t;
00102
00103 // 7.18.1.3 Fastest minimum-width integer types
00104 typedef int8_t    int_fast8_t;
00105 typedef int16_t   int_fast16_t;
00106 typedef int32_t   int_fast32_t;
00107 typedef int64_t   int_fast64_t;
00108 typedef uint8_t   uint_fast8_t;
00109 typedef uint16_t  uint_fast16_t;
00110 typedef uint32_t  uint_fast32_t;
00111 typedef uint64_t  uint_fast64_t;
00112
00113 // 7.18.1.4 Integer types capable of holding object pointers
00114 #ifdef _WIN64 // [
00115    typedef signed __int64    intptr_t;
00116    typedef unsigned __int64  uintptr_t;
00117 #else // _WIN64 ][
00118    typedef _W64 signed int   intptr_t;
00119    typedef _W64 unsigned int uintptr_t;
00120 #endif // _WIN64 ]
00121
00122 // 7.18.1.5 Greatest-width integer types
00123 typedef int64_t   intmax_t;
00124 typedef uint64_t  uintmax_t;
00125
00126
00127 // 7.18.2 Limits of specified-width integer types
00128
00129 #if !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS) // [   See footnote 220 at page 257 and footnote
       221 at page 259
00130
00131 // 7.18.2.1 Limits of exact-width integer types
00132 #define INT8_MIN     ((int8_t)_I8_MIN)
00133 #define INT8_MAX     _I8_MAX
00134 #define INT16_MIN    ((int16_t)_I16_MIN)
00135 #define INT16_MAX    _I16_MAX
00136 #define INT32_MIN    ((int32_t)_I32_MIN)
00137 #define INT32_MAX    _I32_MAX
00138 #define INT64_MIN    ((int64_t)_I64_MIN)
00139 #define INT64_MAX    _I64_MAX
00140 #define UINT8_MAX    _UI8_MAX
00141 #define UINT16_MAX   _UI16_MAX
00142 #define UINT32_MAX   _UI32_MAX
00143 #define UINT64_MAX   _UI64_MAX
00144
00145 // 7.18.2.2 Limits of minimum-width integer types
00146 #define INT_LEAST8_MIN    INT8_MIN
00147 #define INT_LEAST8_MAX    INT8_MAX
00148 #define INT_LEAST16_MIN   INT16_MIN
00149 #define INT_LEAST16_MAX   INT16_MAX
00150 #define INT_LEAST32_MIN   INT32_MIN
00151 #define INT_LEAST32_MAX   INT32_MAX
00152 #define INT_LEAST64_MIN   INT64_MIN
00153 #define INT_LEAST64_MAX   INT64_MAX
00154 #define UINT_LEAST8_MAX   UINT8_MAX
00155 #define UINT_LEAST16_MAX  UINT16_MAX
00156 #define UINT_LEAST32_MAX  UINT32_MAX
00157 #define UINT_LEAST64_MAX  UINT64_MAX
00158
00159 // 7.18.2.3 Limits of fastest minimum-width integer types
00160 #define INT_FAST8_MIN    INT8_MIN
00161 #define INT_FAST8_MAX    INT8_MAX
00162 #define INT_FAST16_MIN   INT16_MIN
00163 #define INT_FAST16_MAX   INT16_MAX
00164 #define INT_FAST32_MIN   INT32_MIN
00165 #define INT_FAST32_MAX   INT32_MAX
00166 #define INT_FAST64_MIN   INT64_MIN
00167 #define INT_FAST64_MAX   INT64_MAX
00168 #define UINT_FAST8_MAX   UINT8_MAX
00169 #define UINT_FAST16_MAX  UINT16_MAX
00170 #define UINT_FAST32_MAX  UINT32_MAX
00171 #define UINT_FAST64_MAX  UINT64_MAX
```

```
00172
00173 // 7.18.2.4 Limits of integer types capable of holding object pointers
00174 #ifdef _WIN64 // [
00175 #  define INTPTR_MIN   INT64_MIN
00176 #  define INTPTR_MAX   INT64_MAX
00177 #  define UINTPTR_MAX  UINT64_MAX
00178 #else // _WIN64 ][
00179 #  define INTPTR_MIN   INT32_MIN
00180 #  define INTPTR_MAX   INT32_MAX
00181 #  define UINTPTR_MAX  UINT32_MAX
00182 #endif // _WIN64 ]
00183
00184 // 7.18.2.5 Limits of greatest-width integer types
00185 #define INTMAX_MIN   INT64_MIN
00186 #define INTMAX_MAX   INT64_MAX
00187 #define UINTMAX_MAX  UINT64_MAX
00188
00189 // 7.18.3 Limits of other integer types
00190
00191 #ifdef _WIN64 // [
00192 #  define PTRDIFF_MIN  _I64_MIN
00193 #  define PTRDIFF_MAX  _I64_MAX
00194 #else  // _WIN64 ][
00195 #  define PTRDIFF_MIN  _I32_MIN
00196 #  define PTRDIFF_MAX  _I32_MAX
00197 #endif  // _WIN64 ]
00198
00199 #define SIG_ATOMIC_MIN  INT_MIN
00200 #define SIG_ATOMIC_MAX  INT_MAX
00201
00202 #ifndef SIZE_MAX // [
00203 #  ifdef _WIN64 // [
00204 #     define SIZE_MAX  _UI64_MAX
00205 #  else // _WIN64 ][
00206 #     define SIZE_MAX  _UI32_MAX
00207 #  endif // _WIN64 ]
00208 #endif // SIZE_MAX ]
00209
00210 // WCHAR_MIN and WCHAR_MAX are also defined in <wchar.h>
00211 #ifndef WCHAR_MIN // [
00212 #  define WCHAR_MIN  0
00213 #endif  // WCHAR_MIN ]
00214 #ifndef WCHAR_MAX // [
00215 #  define WCHAR_MAX  _UI16_MAX
00216 #endif  // WCHAR_MAX ]
00217
00218 #define WINT_MIN  0
00219 #define WINT_MAX  _UI16_MAX
00220
00221 #endif // __STDC_LIMIT_MACROS ]
00222
00223
00224 // 7.18.4 Limits of other integer types
00225
00226 #if !defined(__cplusplus) || defined(__STDC_CONSTANT_MACROS) // [   See footnote 224 at page 260
00227
00228 // 7.18.4.1 Macros for minimum-width integer constants
00229
00230 #define INT8_C(val)  val##i8
00231 #define INT16_C(val) val##i16
00232 #define INT32_C(val) val##i32
00233 #define INT64_C(val) val##i64
00234
00235 #define UINT8_C(val)  val##ui8
00236 #define UINT16_C(val) val##ui16
00237 #define UINT32_C(val) val##ui32
00238 #define UINT64_C(val) val##ui64
00239
00240 // 7.18.4.2 Macros for greatest-width integer constants
00241 #ifndef INTMAX_C
00242 #  define INTMAX_C   INT64_C
00243 #endif
00244
00245 #ifndef UINTMAX_C
00246 #  define UINTMAX_C  UINT64_C
00247 #endif
00248
00249 #endif // __STDC_CONSTANT_MACROS ]
00250
00251
00252 #endif // _MSC_STDINT_H_ ]
```