Quantitation - a library of Uncertain classes

Paul A. Bristow

Table of Contents

Preface

Preface

This library provides a collection of classes and functions for handling uncertain reals and measurements.

Uncertainty as standard deviation and degrees of freedom are propagated through calculations using the uncertain type, and are input and output including the uncertainty estimates typically from standard deviation and degrees of freedom, usually the number of observations less one.

See Boost.Quan HTML Manual.

and/or equivalent PDF Manual

Examples are in folder (and subfolders): examples and C++ include files are in folder (and subfolders):

https://github.com/pabristow/quan/tree/main/include/boost/quan

History and status

Status

Important

This is NOT an official Boost library, but remains a library under construction and enhancement the code is functional, but interfaces, library structure, and function and distribution names may still be changed without notice.

Note

Comments and suggestions (even bugs!) to Paul.A.Bristow (at) hetp (dot) u-net (dot) com or preferably raise a Github issue Github repository Quan library.

It was proposed for Boost as a partner of the Github repository SVG_plot library , but rejected before review as being out-of-scope for Boost.

History

The library was initially informed by a post of a C++ class for uncertain values by Evan Manning in 1996.

Over the years since it has been enhanced, in particular to handle measurements and observations of uncertain data, and then to use the display features with the Github repository SVG_plot library to show not just mathematical functions, but plots showing uncertainties, boxplots and confidence intervals or confidence ellipses.

In 2021, a major refactoring and reburbishment of both documentation and the code has been undertaken. Numerous bugs have been squashed, new features added and more examples added. This has permitted improvements in the Github repository SVG_plot library that uses Github repository Quan library to label data-points.

How To Use This Documentation

- Tutorials are listed in the Table of Contents and include many examples that should help you get started quickly.
- Source code of the many Examples will often be your quickest option. They often deliberately use many features, often producing examples of outstandingly bad taste!
- Reference section prepared using Doxygen will help fine-tuning the appearance of your graphs.
- Several indexes (especially the function index) will also help you find which of the several hundred options you need.
- If you have a feature request, or if it appears that the implementation is in error, please check the TODO page first, as well as the rationale section.

If you do not find your idea/complaint, email me direct at pbristow (at) hetp (dot) u-net (dot) com, or better by posting a Github issue at Github repository Quan library.

Admonishments

Note

These blocks typically go into more detail about an explanation given above.

Tip

These blocks contain information that you may find helpful while coding.

Important

These contain information that is imperative to understanding a concept. Failure to follow suggestions in these blocks will probably result in undesired behavior. Read all of these you find.

Warning

It is imperative that you follow these. Failure to do so will lead to incorrect, and very likely undesired, results.

Getting started

Windows

If you not have Boost installed already, start with Getting started on Windows (and Mingw, Mingw-w64, Msys, Cygwin) but stop at the end of section 3: Header-Only Libraries.

In order to use uncertain and measure classes, there are NO libraries to be built - it is header-only.

But you do need to identify your Boost root location. Something like X:my_boost/ as explained in section 2. You might have this saved in a variable $BOOST_ROOT.

You can obtain a zip or clone from Github repository Quan library. The download or clone of the quan library from Github repository Quan library - you can put wherever you like. It fits neatly into the Boost filestructure tree, but can be elsewhere provided that the include path for the compiler is specified. (Placing in a different partition (different drive letter) may cause difficulties).

If using Visual Studio, the include folder for your project can be set thus: Configuration Properties, VC++ Directories, Include directories, X:/boost ($BOOST_ROOT) and also add X:/svg_plot/include/boost/quan for the location of the Quan library.

If using CodeBlocks and similar IDEs, for your Project, Build Options... Search directories tab, Add and chose folder containing Quan /include.

If you are using Boost's B2/bjam documentation, then you need to ensure that your jamfile (system, user or project) contains an <include> <*path-to-quan*>.

See an example in Jamfile.v2 containing an include folder specified the BOOST_ROOT

```
<include>../../..
```

tests and examples can be run by listing in the folder's `Jamfile.v2` thus

```
test-suite "Quan and Unc library examples"
  :
    [ run quan_simple.cpp : : : [ requires cxx11_numeric_limits ] ]
    [ run quan_complex.cpp : : : [ requires cxx11_numeric_limits ] ]
    # ...  more targets to run
   [ exe quan_executable.cpp : : : [ requires cxx11_numeric_limits ] ]   # (or just build quan_ex↲
ecutable.exe)
  ;
```

To run all these, cd to the folder containing the jamfile and type

```
b2
```

This will display its output to the console. You will probably want to send to a log file thus:

```
b2 > quan_stuff_1apr2022.log
```

This will build all the targets in the jamfile that need (re-)building. If you want to build just one target, specify the target name

```
b2 quan_simple > quan_stuff_1apr2022.log
```

Note

The target name is normally just the filename **without** the file type, and not quan_simple.cpp.

After going through the 'edit-compile-run-crash-cycle' for a single troublesome target, you will also want to (re-)build all the targets with the -a option.

```
b2 -a > all_quan_stuff_1apr2022.log
```

There are no additional linker options required by the quan library.

You will need to select options to use C++14 (the current default C++ Standard for Visual Studio 2019 Community).

And finally, when writing your examples using Quan library, do not forget that all the Quan features are in `namespace boost::quan` so you will need to either specify the entire namespace `using namespace boost::quan;` or individual items that you use specifically, for example: `using boost::quan::unc;`. Use of a private namespace should prevent C++ type name collisions and/or name ambiguities.

*Nix Installation

Most of the procedure is the same as for Windows 10.

Uncertain Classes for Propagation of Uncertainties according to a pure Gaussian model.

This library was originally based on code by Evan Manning (manning@alumni.caltech.edu) Evan Marshal Manning, C/C++ Users Journal, March 1996 page 29 to 38. original downloaded from ftp://beowulf.jpl.nasa.gov/pub/manning.

This is a simple model of uncertainties, designed to accompany an article published in C/C++ Users Journal March 1996 at C/C++ Users Journal March 1996.

(A fuller collection of even fancier classes also given in Evan Marshal Manning's code in `unc.h`).

The key benefit of using an uncertain class is that all uncertainties are properly propagated through calculations.

Other information about the observation or measurement standard deviation & its uncertainty, for example, degrees of freedom, were added by Paul A. Bristow in Mar 1998, and have been refined further since then.

A simple example is construction of an uncertain value with a best estimated value of 1.23, and uncertainty (standard deviation) of 0.45, being the result of 10 observations (so 9 degrees of freedom):

```
uncun u(1.23, 0.45F, 9);
```

and output using `operator<<` adding uncertainty, +/- confidence interval <in angle brackets> and degrees of freedom (in round brackets):

```
std::cout << "URealCorr u(1.23, 0.45, 9); = " << plusminus << addlimits << adddeg↲
free << u << std::endl;
```

to display

```
URealCorr u(1.23, 0.45F, 9) = 1.2 +/-0.45 <0.94, 1.52> (9)
```

Tip

Nearly all examples use chaining of functions to set options and to chain `operator<<` and `operator>>`.

The extensive output features have been informed by "Uncertainty as Applied to Measurements and Calculations", John Denker, http://www.av8n.com/physics/uncertainty.htm#sec-crank3.

Some simple rules that apply whenever displaying a number:

1. Use many enough digits to avoid unintended loss of information.

2. Use few enough digits to be reasonably convenient.

3. Avoid excessive 'noisy' digits that imply unwarranted precision.

4. For other calculations, keep all potentially significant digits (C++ std::numeric_limits<>::max_digits10, 17 for type double).

An additional objective was to avoid using **more** than enough digits to avoid clutter, and more importantly, to avoid giving an impression of more precision or accuracy than the underlying data implies.

There are far too many examples in real life of entirely unwarranted decimal digits, for example, quoting a value of 94% when in fact, there is an uncertainty of 10%, an important uncertainty obscured by the digit 4, when the 'true' value might lie anywhere between 80% and 100%. A more informative representation would be 90%.

The term 'uncertainty' has been used in preference to the more common word 'error' and 'error bars' to express confidence intervals. **This chosen mainly because there are no errors involved! - only a lack of certainty.** There are many possible values that approximate the 'true' value (if such a 'true' thing exists).

So the software provides options to display not only the 'best' or 'mean' or 'central' estimate, but also the uncertainty, degrees of freedom and confidence intervals taking the distribution of the observations. Distributions of types 'normal' or Gaussian (common noisy observations), triangular, and uniform are considered.

Values that are **exact** by definition, for example, the inch-mm conversion factor is exactly 2.54. A value like 2.54 cannot be represented exactly as a floating-point type `double`, so it is useful to be able to distinguish any computational uncertainty machine epsilon (2[super -53] ≅ 1.1e-16) from real uncertainty. Conversely exact integral values should be distinguishable from real observations that happen to have a integral value. To achieve this, specialized constructors are provided for both `int` and `double`.

Many measurements suffer from quantization (uncertainty from analog-to-digital convertors that have too few bits of resolution above the random noise level). To avoid quantization of values read in by other system, it is also possible to ensure that potentially noisy random digits can be output adding at least one extra digit for the mean value.

Boost.Quan library C++ Reference

Header <boost/quan/meas.hpp>

Class `Meas` for measurements using an uncertain class `unc`.

Uses a measurement or observation with information about uncertainty, mainly as a standard deviation, recorded in an uncertain type Unc and also including measurement details order number, &/or time-stamp.

```
namespace boost {
  namespace quan {
    class Meas;

    typedef double real_type;
    std::ostream & operator<<(std::ostream &, const Meas &);
    std::ostream & operator<<(std::ostream &, const std::pair< Meas, Meas > &);
    std::istream & operator>>(std::istream & is, Meas & p);
    template<> double unc_of(Meas);
    template<> std::pair< double, double > uncs_of(std::pair< Meas, uncun >);
    template<> double value_of(Meas);
    template<> std::pair< double, double > values_of(std::pair< Meas, uncun >);
  }
}
```

Class Meas

boost::quan::Meas — Class for Measurement of an value stored as an uncertain type Uncun and also including some ID, and order of measurement, and/or time-date stamp.

Synopsis

```
// In header: <boost/quan/meas.hpp>


class Meas : public boost::quan::unc< false > {
public:
  // construct/copy/destruct
  Meas(const Meas &);
  Meas(double const = 0.);
  Meas(uncun, std::string = "",
       boost::posix_time::ptime = (boost::posix_time::not_a_date_time),
       int = -1);
  Meas & operator=(const Meas &);
  ~Meas();

  // friend functions
  std::ostream & operator<<(std::ostream &, const Meas &);

  // public member functions
  Meas & abs(const Meas &);
  Meas & abs(Meas &);
  unsigned short int deg_free() const;
  void deg_free(short unsigned int);
  unsigned short int degFree(void) const;
  float deviation(void) const;
  std::string & id();
  void id(std::string &);
  double mean(void) const;
  operator value_type() const;
  bool operator!=(const Meas &) const;
  bool operator!=(const unc< is_correlated > &) const;
  unc< is_correlated > & operator*=(const double &);
  unc< is_correlated > & operator*=(const unc< is_correlated > &);
  Meas operator+(const Meas &) const;
  unc< is_correlated > operator+(void) const;
  unc< is_correlated > operator++(int);
  unc< is_correlated > operator++(void);
  unc< is_correlated > & operator+=(const double &);
  unc< is_correlated > & operator+=(const unc< is_correlated > &);
  Meas operator-(void);
  Meas operator-(void) const;
  unc< is_correlated > operator--(int);
  unc< is_correlated > operator--(void);
  unc< is_correlated > & operator-=(const double &);
  unc< is_correlated > & operator-=(const unc< is_correlated > &);
  unc< is_correlated > & operator/=(const double &);
  unc< is_correlated > & operator/=(const unc< is_correlated > &);
  bool operator<(const Meas &) const;
  bool operator<(const unc< is_correlated > &) const;
  bool operator==(const Meas &) const;
  bool operator==(const unc< is_correlated > &) const;
  bool operator>(const Meas &) const;
  int order();
  void order(int);
  void setUncTypes(short unsigned int);
  float std_dev() const;
  void std_dev(float);
  boost::posix_time::ptime & time();
  void time(boost::posix_time::ptime &);
  unsigned short int types() const;
  void types(short unsigned int);
  unsigned short int uncFlags(void) const;
  double value() const;
  void value(double);

  // public static functions
  static bool earlier(const Meas &, const Meas &);
  static bool equal_toUnc(const Meas &, const Meas &);
  static bool equalU(const unc< is_correlated > &,
                     const unc< is_correlated > &);
  static bool equalU2(const unc< is_correlated > &,
                      const unc< is_correlated > &);
  static bool greater2U(const Meas &, const Meas &);
  static bool greaterU1(const Meas &, const Meas &);
  static bool less(const Meas &, const Meas &);
  static bool less2U(const Meas &, const Meas &);
  static bool lessAbsM(const Meas &, const Meas &);
  static bool lessU(const unc< is_correlated > &,
                    const unc< is_correlated > &);
  static bool lessU1(const Meas &, const Meas &);
  static bool lessU2(const unc< is_correlated > &,
                     const unc< is_correlated > &);
  static bool moreU(const unc< is_correlated > &,
                    const unc< is_correlated > &);
  static bool precedes(const Meas &, const Meas &);

  // public data members
  std::string id_;  // Meas Member variables. Time and order values could be: 1 no known order. ↵
2 known order, but no times. 3 times, from which order can be calculated, or both order and ↵
times given, so may need to be checked for consistency.
  int order_;  // Index from 0 (or 1 perhaps?) -1 == unknown.
  boost::posix_time::ptime time_;  // Posix time from Boost.Date_time.
};
```

Description

Meas public construct/copy/destruct

1.
```
Meas(const Meas &);
```

2.
```
Meas(double const d = 0.);
```

3.
```
Meas(uncun u, std::string id = "",
     boost::posix_time::ptime ti = (boost::posix_time::not_a_date_time),
     int o = -1);
```

4.
```
Meas & operator=(const Meas & rhs);
```

   Assignment operator.

5.
```
~Meas();
```

`Meas` friend functions

1.
   ```
   std::ostream & operator<<(std::ostream &, const Meas &);
   ```

   Extract operator to output items of class `Meas`.

`Meas` public member functions

1.
   ```
   Meas & abs(const Meas & rhs);
   ```

2.
   ```
   Meas & abs(Meas & rhs);
   ```

3.
   ```
   unsigned short int deg_free() const;
   ```

   Returns: Estimate of number of degrees of freedom, usually = number of observations -1.

4.
   ```
   void deg_free(short unsigned int df);
   ```

   Parameters: `df` Number ofdegrees of freedom, usually = number of observations -1, so 0 = means just one observation.

5.
   ```
   unsigned short int degFree(void) const;
   ```

6.
   ```
   float deviation(void) const;
   ```

7.
   ```
   std::string & id();
   ```

   Returns: a identication `std::string` describing the measurement.

8.
   ```
   void id(std::string & ident);
   ```

   Set a identication string describing the measurement.

   Parameters: `ident` as a @ std::string

9.
   ```
   double mean(void) const;
   ```

10.
    ```
    operator value_type() const;
    ```

11.
    ```
    bool operator!=(const Meas & p) const;
    ```

12.
    ```
    bool operator!=(const unc< is_correlated > & x) const;
    ```

13.
```
unc< is_correlated > & operator*=(const double & a);
```

14.
```
unc< is_correlated > & operator*=(const unc< is_correlated > & ud);
```

15.
```
Meas operator+(const Meas &) const;
```

16.
```
unc< is_correlated > operator+(void) const;
```

17.
```
unc< is_correlated > operator++(int);
```

18.
```
unc< is_correlated > operator++(void);
```

19.
```
unc< is_correlated > & operator+=(const double & a);
```

20.
```
unc< is_correlated > & operator+=(const unc< is_correlated > & ud);
```

21.
```
Meas operator-(void);
```

22.
```
Meas operator-(void) const;
```

23.
```
unc< is_correlated > operator--(int);
```

24.
```
unc< is_correlated > operator--(void);
```

25.
```
unc< is_correlated > & operator-=(const double & a);
```

26.
```
unc< is_correlated > & operator-=(const unc< is_correlated > & ud);
```

27.
```
unc< is_correlated > & operator/=(const double & a);
```

operator /=

Note

Dividing by constant doesn't change degrees of freedom or uncertainty. All indicators & restrictions on value removed. Must assume double is irrational and not integer, NOT int/int becomes rational as operator /= .

28.
```
unc< is_correlated > & operator/=(const unc< is_correlated > & ud);
```

29.
```
bool operator<(const Meas & rhs) const;
```

30.
```
bool operator<(const unc< is_correlated > & x) const;
```

Predicate compare operator< for use by std::sort etc. (Note const needed to use with less!)

31.
```
bool operator==(const Meas & p) const;
```

32.
```
bool operator==(const unc< is_correlated > & x) const;
```

33.
```
bool operator>(const Meas & rhs) const;
```

34.
```
int order();
```

Returns: Order of this measurement in a sequence of measurements, often at fixed intervals.

35.
```
void order(int order_no);
```

Set the order of this measurement in a sequence of measurements, often at fixed intervals.

36.
```
void setUncTypes(short unsigned int type);
```

Parameters: `type` bits indicating type of uncertain real.

37.
```
float std_dev() const;
```

Get estimate of uncertainty of value of uncertain type.

Returns: Estimate of uncertainty as standard deviation of value of uncertain type.

38.
```
void std_dev(float unc);
```

Set estimate of uncertainty of value of uncertain type.

Parameters: `unc` Estimate of uncertainty as standard deviation of value of uncertain type.

39.
```
boost::posix_time::ptime & time();
```

Returns: a time_point as ptime when the measurement was made.

40.
```
void time(boost::posix_time::ptime & tim);
```

Set a time_point as ptime when the measurement was made.

41.
```
unsigned short int types() const;
```

Returns: Types of uncertain real, encoded as a bitmap.

42.
```
void types(short unsigned int type);
```

43.
```
unsigned short int uncFlags(void) const;
```

44.
```
double value() const;
```

Get central estimate of value of uncertain type.

Returns: Central 'best' estimate of value of uncertain type.

45.
```
void value(double value);
```

Set central estimate of value of uncertain type.

Parameters: `value` Central estimate of value of uncertain type.

`Meas` public static functions

1.
```
static bool earlier(const Meas & l, const Meas & r);
```

2.
```
static bool equal_toUnc(const Meas & l, const Meas & r);
```

3.
```
static bool equalU(const unc< is_correlated > & l,
                   const unc< is_correlated > & r);
```

4.
```
static bool equalU2(const unc< is_correlated > & l,
                    const unc< is_correlated > & r);
```

5.
```
static bool greater2U(const Meas & l, const Meas & r);
```

6.
```
static bool greaterU1(const Meas & l, const Meas & r);
```

Returns: `true` if

7.
```
static bool less(const Meas & l, const Meas & r);
```

8.
```
static bool less2U(const Meas & l, const Meas & r);
```

9.
```
static bool lessAbsM(const Meas & l, const Meas & r);
```

10.
```
static bool lessU(const unc< is_correlated > & l,
                  const unc< is_correlated > & r);
```

Compare two uncertain values.

Parameters:	`l`	Uncertain value.
	`r`	Uncertain value.
Returns:	true if l is effectively less than r.	

11.
```
static bool lessU1(const Meas & l, const Meas & r);
```

12.
```
static bool lessU2(const unc< is_correlated > & l,
                   const unc< is_correlated > & r);
```

13.
```
static bool moreU(const unc< is_correlated > & l,
                  const unc< is_correlated > & r);
```

14.
```
static bool precedes(const Meas & l, const Meas & r);
```

`Meas` public public data members

1.
```
std::string id_;
```

Identification info, if any, else "".

Function operator<<

boost::quan::operator<< — Extract operator for output of items of class Meas.

Synopsis

```
// In header: <boost/quan/meas.hpp>


std::ostream & operator<<(std::ostream & os, const Meas & p);
```

Description

Extract operator to output items of class `Meas`.

Function operator<<

boost::quan::operator<<

Synopsis

```
// In header: <boost/quan/meas.hpp>


std::ostream &
operator<<(std::ostream & os, const std::pair< Meas, Meas > & mp);
```

Description

Output a pair (X and Y) of uncertain measurement values with (if defined) uncertainty and degrees of freedom etc

For example: "1.23 +/- 0.01 (13), 3.45 +/- 0.06 (78)".

Function template unc_of

boost::quan::unc_of

Synopsis

```
// In header: <boost/quan/meas.hpp>


template<> double unc_of(Meas m);
```

Description

Returns:	uncertainty of value as Meas.std_dev() as a `double`.

Function template uncs_of

boost::quan::uncs_of — Get uncertainties (standard deviation) of a pair of values.

Synopsis

```
// In header: <boost/quan/meas.hpp>


template<> std::pair< double, double > uncs_of(std::pair< Meas, uncun > vp);
```

Description

Function template value_of

boost::quan::value_of

Synopsis

```
// In header: <boost/quan/meas.hpp>


template<> double value_of(Meas v);
```

Description

Returns:	Meas.value() as a `double`.
Returns:	Meas.value() as a double.

Function template values_of

boost::quan::values_of — Get values of a pair of values.

Synopsis

```
// In header: <boost/quan/meas.hpp>


template<> std::pair< double, double > values_of(std::pair< Meas, uncun > vp);
```

Description

Header <boost/quan/pair_io.hpp>

Provide operator<< to output pair, surrounded by < > and separated by comma, for example: "<1.23, 4.56>". Typically used to show confidence intervals around a mean.

Polite to keep this a private detail namespace to avoid clash with other implementations!

Paul A. Bristow

Oct 2009, 2012, 2021

```


// Output pair, separated by comma & space, enclosed by <> and separated by comma & space.
template<typename charT, typename traits>
  std::basic_ostream< charT, traits > &
  operator<<(std::basic_ostream< charT, traits > & os,
             const std::pair< double, double > & p);

// Output both items of a pair of items (not necessarily the same type), enclosed by <> and sep↵
arated by comma & space.
template<typename charT, typename traits, typename T1, typename T2>
  std::basic_ostream< charT, traits > &
  operator<<(std::basic_ostream< charT, traits > & os,
             const std::pair< T1, T2 > & p);
```

Header <boost/quan/rounding.hpp>

Common rounding and 'proper' rounding.

http://www.diycalculator.com/popup-m-round.shtml#A3 all rounding types, including round-to-half and asymmetric and symmetric versions.
http://www.chem1.com/acad/webtext/pre/mm3.html "The purpose in rounding off is to avoid expressing a value to a greater degree of precision than is consistent with the uncertainty in the measurement." "Observed values should be rounded off to the number of digits that most accurately conveys the uncertainty in the measurement."

```
namespace boost {
  namespace quan {

    enum distribution_type { gaussian = = 0, uniform = = 1, triangular = = 2,
                             undefined = = 3 };

    const unsigned int maxdigits10;
    static const double oneDivSqrtSix;
    static const double oneDivTwoSqrtSix;
    bool scaled;
    static const double sqrt_2;
    static const double sqrt_3;
    static const double sqrt_6;
     BOOST_STATIC_ASSERT(std::numeric_limits< double >::is_iec559);
     BOOST_STATIC_ASSERT(std::numeric_limits< double >::is_specialized);
    double cdf_tri(double);
    double cdf_uni(double);
    std::pair< double, double >
    conf_interval(double, double, double = 1., double = 0.05,
                  distribution_type = gaussian);
    double delta(double, double, distribution_type = gaussian);
    std::ostream & operator<<(std::ostream &, std::pair< double, double > &);
    void out_confidence_interval(std::pair< double, double >, int,
                                 std::ostream & = std::cout);
    void out_value_df_limits(double, double, int = 1,
                             std::ostream & = std::cout);
    void out_value_limits(double, double, std::pair< double, double >, int,
                          std::ostream & = std::cout);
    double quantile_tri(double);
    double quantile_uni(double);
    double round_1(double);
    double round_2(double);
    double round_3(double);
    template<typename FPT> std::string round_e(FPT, int);
    template<typename FPT> std::string round_f(FPT, int);
    int round_m(double = 0.01, double = 0., unsigned int = 2U,
                distribution_type = gaussian);
    template<typename FPT> std::string round_ms(FPT, signed int);
    double round_nth(double, unsigned int);
    template<typename FPT> FPT round_sig(FPT, int);
    template<typename FPT> FPT round_to_n(FPT, int);
    std::string round_ue(double, double, double = 0.01, unsigned int = 2U);
    double rounded_div_value(double, double);
  }
}
```

Global maxdigits10

boost::quan::maxdigits10

Synopsis

```
// In header: <boost/quan/rounding.hpp>

const unsigned int maxdigits10;
```

Global oneDivSqrtSix

boost::quan::oneDivSqrtSix

Synopsis

```
// In header: <boost/quan/rounding.hpp>

static const double oneDivSqrtSix;
```

Global oneDivTwoSqrtSix

boost::quan::oneDivTwoSqrtSix

Synopsis

```
// In header: <boost/quan/rounding.hpp>

static const double oneDivTwoSqrtSix;
```

Global scaled

boost::quan::scaled

Synopsis

```
// In header: <boost/quan/rounding.hpp>

bool scaled;
```

Global sqrt_2

boost::quan::sqrt_2

Synopsis

```
// In header: <boost/quan/rounding.hpp>

static const double sqrt_2;
```

Global sqrt_3

boost::quan::sqrt_3

Synopsis

```
// In header: <boost/quan/rounding.hpp>

static const double sqrt_3;
```

Global sqrt_6

boost::quan::sqrt_6

Synopsis

```
// In header: <boost/quan/rounding.hpp>

static const double sqrt_6;
```

Function cdf_tri

boost::quan::cdf_tri

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double cdf_tri(double z);
```

Description

Cumulative Distribution Function (CDF) of Triangular distribution. Gejza Wimmer, Viktor Witkovsky, Tomas Duby, Proper rounding of the measurement result under the assumption of triangular distribution, Measurement Science Review, Vol 2, section 1, (2002), page 24, equation 7.

Parameters:	z	Ramdom variate.
Returns:	CDF of triangular distribution.	

Function cdf_uni

boost::quan::cdf_uni

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double cdf_uni(double z);
```

Description

Cumulative Distribution Function (CDF) of Uniform distribution Un(-sqrt_3, + sqrt_3), with an expectation of mean zero and variance unity. Gejza Wimmer & Victor Witkovsky, Measurement Science Review, volume 2 section 1 2002, page 3, equation 5.

Returns:	Cumulative distribution Function of uniform distribution.

Function conf_interval

boost::quan::conf_interval

Synopsis

```
// In header: <boost/quan/rounding.hpp>


std::pair< double, double >
conf_interval(double mean, double sigma, double df = 1., double alpha = 0.05,
              distribution_type distrib = gaussian);
```

Description

Calculate confidence interval for chosen alpha confidence level and chosen distribution type.

Uses confidence interval equations from:
Gejza Wimmer, Viktor Witkovsky, Tomas Duby Measurement Science and Technolology, 11 (2000) 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X. Proper rounding of the measurement result under the assumption of gaussian distribution,
Confidence interval equation 6, p 1660.

Gejza Wimmer, Viktor Witkovsky, Tomas Duby
Measurement Science and Technolology, 11 (2000) 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X.
Proper rounding of the measurement result under the assumption of triangular distribution,
Measurement Science Review, Vol 2, section 1, (2002), pages 21 to 31.
Confidence interval Equation 12, page 25.
Gejza Wimmer, Viktor Witkovsky,
Proper rounding of the measurement result under the assumption of uniform distribution,
Measurement Science Review, Vol 2, section 1, (2002), pages 1 - 7.
Confidence interval, page 5, equation 13.

Parameters:	`alpha`	Confidence (0. < unc < 1.), typically 0.05 for 95% confidence. alpha/2 = 0.025 quantile = -1.96, and 1 - alpha/2 is 1.96 for normal distribution, so typically confidence interval is roughly twice unc either side (+ or -) of mean.
	`df`	Degrees of freedom (usually number of observations -1).
	`distrib`	Distribution type, Gaussian(default), uniform, triangular, laplace.
	`mean`	Interval estimate for mean mu.
	`sigma`	Standard uncertainty sigma from variance sigma.
Returns:	Confidence interval upper and lower limits as a `std::pair` of `double`.	

Function delta

boost::quan::delta

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double delta(double loss_risk, double rounded_div_value,
             distribution_type distrib = gaussian);
```

Description

Calculate Wimmer delta function using equation 24, p 1664.

Gejza Wimmer, Viktor Witkovsky, Tomas Duby
Measurement Science and Technology, 11 (2000) pages 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X.
Proper rounding of the measurement results under normality assumptions.
Gejza Wimmer, Viktor Witkovsky, Proper rounding of the measurement result under the assumption of uniform distribution, Measurement Science Review, Vol 2, section 1, (2002), pages 1 - 7.
Gejza Wimmer, Viktor Witkovsky, Proper rounding of the measurement result under the assumption of triangular distribution, Measurement Science Review, Vol 2, section 1, (2002), pages 21 to 31.

Note

rounding_loss is called epsilon by Wimmer et all.

Note

`loss_risk` is usually 0.05 (equivalent to 95%), values in range 0.2 to 0.01 make good sense, 0.2 (80%) risks a fair bit of loss from rounding, 0.01 (1%) causes almost no loss.

Parameters:	`distrib`	Distribution type, normal or gaussian(default), uniform, triangular, or laplace.
	`loss_risk`	Proper-rounding increase of confidence-interval because of rounding, must be positive and 'small'.
	`rounded_div_value`	Ratio rounded / unrounded (rounded_div_value is assumed <= 1).
Returns:	Approximation of Wimmer delta function using equation 24, p 1664, for normal distribution, and similar for triangular and uniform. Returns -1 if rounding_loss is too 'tight', so must be increased for delta to be calculated. Throwing an exception might be better here?	

Function operator<<

boost::quan::operator<<

Synopsis

```
// In header: <boost/quan/rounding.hpp>


std::ostream & operator<<(std::ostream & os, std::pair< double, double > & p);
```

Description

Output a pair of `double`s, using < > angle brackets and comma separator, using current stream's precision.

Explicit specialization for `std::pair` for `double` s.
For example:<97.8725, 157.798>

Parameters:	`os`	`std::ostream` for string output.
	`p`	Pair of `double` s.

Function out_confidence_interval

boost::quan::out_confidence_interval

Synopsis

```
// In header: <boost/quan/rounding.hpp>


void out_confidence_interval(std::pair< double, double > ci, int m,
                             std::ostream & os = std::cout);
```

Description

Output to os, confidence interval enclosed in brackets and separated by comma. The precision is controlled by the rounding order `m` used for the mean but with one extra decimal digit of precision, so rounding to m-1 th place. For example: <99.5, 156.5> rounding to -1 position (where the mean was rounded with m == 0, using the units digit to round the tens digit to "128.")

Parameters:		
	`ci`	Confidence interval or limits to be output.
	`m`	Signed position for rounding.
	`os`	`std::ostream` for output.

Function out_value_df_limits

boost::quan::out_value_df_limits

Synopsis

```
// In header: <boost/quan/rounding.hpp>


void out_value_df_limits(double mean, double sigma, int degfree = 1,
                         std::ostream & os = std::cout);
```

Description

Output mean, uncertainty and confidence interval. For example:128. +/- 15 \<99.5, 156.5\>

Parameters:		
	`degfree`	Number of degrees of freedom.
	`mean`	Mean or central estimate of value.
	`os`	`std::ostream` for output.
	`sigma`	Uncertainty estimate as standard deviation.

Function out_value_limits

boost::quan::out_value_limits

Synopsis

```
// In header: <boost/quan/rounding.hpp>


void out_value_limits(double mean, double unc, std::pair< double, double > ci,
                      int m, std::ostream & os = std::cout);
```

Description

Output mean, uncertainty and confidence interval. For example:128. +/- 15 <99.5, 156.5>

Parameters:	`ci`	Confidence interval as an interval.
	`m`	Rounding position.
	`mean`	Mean or central estimate of value.
	`os`	`std::ostream` for output.
	`unc`	Uncertainty estimate as standard deviation.

Function quantile_tri

boost::quan::quantile_tri

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double quantile_tri(double alpha);
```

Description

Quantile or Inverse of cumulative distribution function CDF of triangular distribution. Gejza Wimmer, Viktor Witkovsky, Tomas Duby, Proper rounding of the measurement result under the assumption of triangular distribution, Measurement Science Review, Vol 2, section 1, (2002), page 24, equation 8.

Parameters:	`alpha`	Confidence level.
Returns:	Quantile or Inverse of cumulative distribution function CDF.	

Function quantile_uni

boost::quan::quantile_uni

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double quantile_uni(double alpha);
```

Description

Quantile of uniform distribution Uniform(-sqrt_3, + sqrt_3), expectation or mean zero and variance unity. Wimmer & Witkovsky, Measurement Science Review, volume 2 section 1 2002, page 3, eq 6.

Returns:	Quantile of uniform distribution.

Function round_1

boost::quan::round_1

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double round_1(double v);
```

Description

round value v to 1 digit after decimal point, using 1st digit to round.

Function round_2

boost::quan::round_2

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double round_2(double v);
```

Description

round value v to 2 digit2 after decimal point, using 2nd digit to round.

Function round_3

boost::quan::round_3

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double round_3(double v);
```

Description

round value v to 3 digit2 after decimal point, using 3rd digit to round.

Function template round_e

boost::quan::round_e

Synopsis

```
// In header: <boost/quan/rounding.hpp>


template<typename FPT> std::string round_e(FPT d, int sigdigits);
```

Description

Show only `sigdigits` digits, in minimal scientific or 'e' format only, rounded asymmetric (round-half-up, arithmetic rounding). Round asymmetric avoids values like 0.15, which cannot be stored exactly in binary IEEE 754 floating-point format and are stored internally as 1.499999999999, being rounded unexpectedly to 0.1 rather than 0.2. For example: format -1.2345e12 or 1.2345e-123.

Note

the difference in meaning of precision from default, fixed and scientific, for example: `d = 19.99;`

```
std::cerr << setprecision(2) << d << std::endl; // 20 // rounded
```

```
std::cerr << fixed << setprecision(1) << d << std::endl; // 20.0 - rounded up.
```

```
std::cerr << fixed << setprecision(2) << d << std::endl; // 19.99 - 2 digits after ↵
decimal point.
```

```
std::cerr << scientific << setprecision(2) << d << std::endl; // 2.00e+001 - 2 di↵
gits after decimal point
```

Also remove all redundant exponent characters and digits,

See Also:

http://en.wikipedia.org/wiki/Rounding.

Note

The normal rounding provided by C and C++ is chosen to avoid loss of accuracy and bias in any further **binary** calculations. The rounded decimal digit **strings** returned are provided solely for **human** viewing and should NOT be used for input to other calculations when one or two extra noisy digits are needed to avoid loss of information by rounding. If it is necessary to provide a decimal digit string for input, it should use `std::numeric_limits<FPT>::max_digits10` in C++0x, or 2 + std::numeric_limits<FPT>::digits * 3010/10000 if the platform implementation of C++ Standard IOstream library does not yet provide this. For 64-bit double, `max_digits10` is 17, so write `cout.precision(17)`. This will avoid any bias or loss of accuracy in statistical calculations caused by rounding.

Use the scientific (exponential) format with precision max_digits10 (17 for double).

Logic is using C++ std printf-style rounding to start with a decimal digit string with rounded value of digits10, only the guaranteed correct digits to get the best estimate of the digit to be used for the final rounding. Hopefully, any imprecision caused by inexact presentation is (binary) rounded away.

See Also:

http://www.exploringbinary.com/ for much more detail of defects in other libraries.

Note

Only lowercase letter e is used, and positive exponents are not preceeded by + sign, and leading zeros are omitted. This is done to make the display as compact as possible.
If a fixed width is needed, the user program must provide any necessary padding, using the size (length) of the string.

Parameters:	`d`	value to be converted to decimal digit string.
	`sigdigits`	number of significant digits to show in string.
Returns:	std::string containing value rounded and converted to `sigdigits` decimal digits in exponential format.	

Function template round_f

boost::quan::round_f — Round floating-point value `v` (fixed, not-exponential) to `sigdigits` significant digits. This is variously called 'common rounding', 'round_5_up' or 'proper' rounding.

Synopsis

```
// In header: <boost/quan/rounding.hpp>


template<typename FPT> std::string round_f(FPT v, int sigdigits);
```

Description

Gejza Wimmer, Viktor Witkovsky, Tomas Duby
Measurement Science and Technology, 11 (2000) 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X
Proper rounding of the measurement results under normality assumptions.
Uncertainty of measurement – Part 3: Guide to the expression of uncertainty in measurement (GUM:1995)
ISO Guide 98 (1995) and updated version 2008.

Note

All these functions are templated on floating-point type and are not intended to be called (and thus instantiated) for integral (or other) types. This is because the value will not be output in scientific format, assumed by the code. A check is_floating_point == true is provided.

Note

C++0X provides `std::numeric_limits<double>::max_digits10`; which is maximum number of possibly significant digits, but only `digit10` are guaranteed to be significant, so limit `sigdigits` to `digits10` (15 for IEEE 64-bit double).

•

Parameters:	`sigdigits`	Number of significant digits after rounding.
	`v`	Value to be converted to a decimal digit string, rounded to sigdigits significant digits.
Returns:	`std::string` containing decimal digit string, properly decimally rounded.	

Function round_m

boost::quan::round_m

Synopsis

```
// In header: <boost/quan/rounding.hpp>


int round_m(double rounding_loss = 0.01, double sigma = 0.,
            unsigned int sigma_sigdigits = 2U,
            distribution_type distrib = gaussian);
```

Description

Calculate the Wimmer rounding digit m using delta and rounded_div_value functions p 1661, equation 12.

Measurement Science and Technology, 11 (2000) 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X. Gejza Wimmer, Viktor Witkovsky, Tomas Duby
Proper rounding of the measurement results under normality assumptions.

Parameters:	`distrib`	Type of distribution (default gaussian, or triangular or uniform).
	`rounding_loss`	Rounding loss (as fraction) accepted (default is 0.01 or 1%).
	`sigma`	Uncertainty as standard deviation, for example: 0.01.
	`sigma_sigdigits`	Number of significant decimal digits (default = 2) for uncertainty to be rounded (range 1 to 4, default = 2), depending on degrees of freedom. If degrees_of_freedom <= 2, then 1, else if degrees_of_freedom > 1000, then 3 else default = 2.
Returns:	m Signed position of the digit to be used for rounding, `m == 0` means use the units digit for rounding the tens digit.	

Function template round_ms

boost::quan::round_ms

Synopsis

```
// In header: <boost/quan/rounding.hpp>


template<typename FPT> std::string round_ms(FPT v, signed int m);
```

Description

Round floating-point v (not-exponential) to order m. (m is the index of the roundER digit). This is variously called 'common rounding', 'round_5_up'.

Gejza Wimmer, Viktor Witkovsky, Tomas Duby
Measurement Science and Technology, 11 (2000) 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X
Proper rounding of the measurement results under normality assumptions.
Uncertainty of measurement – Part 3: Guide to the expression of uncertainty in measurement (GUM:1995)
ISO Guide 98 (1995) and updated version 2008.

Note

`m` is the index of the rounder digit, that is the just insignificant digit used to decide if the m+1th digit is to be rounded up or left as is. So `m == 0` means that the roundER digit is the units digits, used to round the tens digit, `m == +1` the roundER is the tens digit, and the rounded digit is the hundreds digit `m == -1` roundER is the tenths digit (0.1), rounding the hundredths digit (0.01).

(Other authors specify the nth digit to be significant, where n = m + 1).

> **Note**
>
> that the most significant sign bit of NaN is recognized by using function signbit, 'sign' of NaNs cannot reliably and portably be tested using "x < 0" because ALL comparisons using NaN are false - by definition.

Parameters:	`m`	Signed digit position to round.
	`v`	Value to convert to decimal digit string after rounding.
Returns:	`std::string` decimal digit string of rounded value.	

Function round_nth

boost::quan::round_nth

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double round_nth(double v, unsigned int d);
```

Description

round value v to d digits after decimal point, using d+1 digit to round.

Function template round_sig

boost::quan::round_sig — Returns v rounded to n significant decimal digits.
http://www.jason.mock.ws/wordpress/2007/02/22/round-a-float-to-of-significant-digits, David A. Pimentel.

Synopsis

```
// In header: <boost/quan/rounding.hpp>


template<typename FPT> FPT round_sig(FPT v, int n);
```

Description

These use log10 and pow and so are vulnerable to binary rounding errors, as the value may not be exactly representable for the type specified. Should give the same value as printf precision. (int)(x < 0 ? x - 0.5 : x + 0.5)) So do we need to do something different for negative doubles? The difference between symmetric and asymmetric rounding?

Parameters:	`n`	number of digificant digits to round to.
	`v`	Floating-point value to be rounded.
Template Parameters:	`FPT`	Floating-point type, for example, fundamental float, double, long
Returns:	Binary representation of decimal rounded value.	

Function template round_to_n

boost::quan::round_to_n

Synopsis

```
// In header: <boost/quan/rounding.hpp>


template<typename FPT> FPT round_to_n(FPT v, int p);
```

Description

Round value v to p decimal digits **after** the decimal point. http://www.jason.mock.ws/wordpress/2007/02/22/round-a-float-to-of-significant-digits

Note

These use log10 and pow and so are vulnerable to binary rounding errors, as the value v may not be exactly representable for the type specified. Should give the same value as printf precision.

Parameters:	`p`	Number of decimal digits **after** the decimal point.
	`v`	Value to round.

Function round_ue

boost::quan::round_ue

Synopsis

```
// In header: <boost/quan/rounding.hpp>


std::string round_ue(double v, double sigma, double loss_risk = 0.01,
                     unsigned int sigdigits = 2U);
```

Description

Properly round value to a decimal-digit `std::string`.

Measurement Science and Technolology, 11 (2000) 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X.
Gejza Wimmer, Viktor Witkovsky, Tomas Duby, Proper rounding of the measurement results under normality assumptions. page 1661, equation 12.

Parameters:	`loss_risk`	Fraction of loss of accuracy from rounding permitted (default 1%).
	`sigdigits`	Number of digits that are significant (default 2).
	`sigma`	Uncertainty of value, usually standard deviation.
	`v`	Central value, often estimate of mean.
Returns:	Decimal digit `std::string` containing properly rounded value as decimal-digits.	

Function rounded_div_value

boost::quan::rounded_div_value

Synopsis

```
// In header: <boost/quan/rounding.hpp>


double rounded_div_value(double rounded, double value);
```

Description

Calculate Wimmer rounded_div_value rounded/unrounded function p 1664.

Measurement Science and Technolology, 11 (2000) 1659-1665. ISSN 0957-0233 S0957-233(00)13838-X. Gejza Wimmer, Viktor Witkovsky, Tomas Duby Proper rounding of the measurement results under normality assumptions.

Parameters:	`rounded`	Value (as `double`) after rounding.
	`value`	Value (as `double`) before rounding. Assumes `rounded < unrounded value`.
Returns:	Ratio rounded / value, a measure of loss of information caused by rounding.	

Header <boost/quan/si_units.hpp>

SI Unit, names and abbreviations & conversion factors.

See NIST 811 Special publication, Barry N Taylor (1995) "Guide for the Use of the International System of Units (SI) & ISO 31, a closely aligned document. NIST version contains more on non-SI units and many useful notes.

```

struct unit;
const unit &
findAnyUnit(const string, unsigned int &, unsigned int &, unsigned int &);
unsigned int
findUnit(const string, const unit *, unsigned int &, unsigned int &);
void showAllNonSIunits(const unit *, ostream &);
void showAllUnits(const unit *, ostream &);
void showNonSIunits(const unit *, ostream &);
void showSIunit(const unit *, ostream &);
void showSIunits(const unit *, ostream &);
```

Struct unit

unit — Fundamental units: Mass, Length, Time, Charge, Temperature, Luminous Intensity & angle.

Synopsis

```
// In header: <boost/quan/si_units.hpp>


struct unit {

  // public data members
  const char * SIunitAbbrev;
  const char ** SIunitNames;
  unsigned char unitAbbrevCount;
  const int * unitLengths;
  const char *** unitNames;
  unsigned char unitNamesCount;
  const char * unitOf;
  unsigned char unitSINameCount;
  unsigned char unitsNamesCount;
  const double * unitToSIfactors;
};
```

Header <boost/quan/type_erasure_printer.hpp>

```

struct _iter;
struct _os;
struct _t;

template<typename CharT = char, typename Traits = std::char_traits<char> >
  class abstract_printer;

template<typename T, typename U = _self> struct base_and_derived;

class decor_printer;namespace boost {
  namespace type_erasure {
    template<typename T, typename U, typename Base>
      struct concept_interface<base_and_derived< T, U >, Base, U>;
    template<typename Os, typename T1, typename T2>
      struct ostreamable<Os, std::pair< T1, T2 >>;
    template<typename T1, typename T2>
      std::ostream & operator<<(std::ostream &, const std::pair< T1, T2 > &);
  }
}
```

Struct _iter

_iter

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>


struct _iter : public boost::type_erasure::placeholder {
};
```

Struct _os

_os

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>


struct _os : public boost::type_erasure::placeholder {
};
```

Struct _t

_t — Placeholder used by the abstract printer during specification of requirements.

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>


struct _t : public boost::type_erasure::placeholder {
};
```

Class template abstract_printer

abstract_printer

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename CharT = char, typename Traits = std::char_traits<char> >
class abstract_printer {
public:
  // types
  typedef boost::type_erasure::any< requirements, _iter > iterator_type;

  // construct/copy/destruct
  ~abstract_printer();

  // public member functions
  template<typename Range, typename CharT, typename Traits>
    void print(const Range &,
               std::basic_ostream< CharT, Traits > & = std::cout) const;

  // protected member functions
  virtual void do_print(iterator_type, iterator_type, ostream_type) const = 0;
};
```

Description

An abstract sequence printer - a 'template' that is inherited to implement the examples of actual printers defined below.

`abstract_printer` public construct/copy/destruct

1.
```
~abstract_printer();
```

`abstract_printer` public member functions

1.
```
template<typename Range, typename CharT, typename Traits> 
  void print(const Range & r, 
             std::basic_ostream< CharT, Traits > & os = std::cout) const;
```

Parameters:	`os`	`std::ostream` for printing, default, for example: std::cout.
	`r`	Container to be printed, for example, a C array, std::vector, std::list ...
Template Parameters:	`Range`	must be a Forward Range whose elements can be printed to `std::ostream` os.

`abstract_printer` protected member functions

1.
```
virtual void 
do_print(iterator_type first, iterator_type last, ostream_type os) const = 0;
```

Declare the pure virtual function `do_print` that is defined in all real printers.

Struct template base_and_derived

base_and_derived — base_and_derived<std::ios_base, _os>,

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename T, typename U = _self> 
struct base_and_derived {

  // public static functions
  static T & apply(U &);
};
```

Description

_os must derive from std::ios_base.

`base_and_derived` public static functions

1.
```
static T & apply(U & arg);
```

Class decor_printer

decor_printer

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>


class decor_printer : public abstract_printer<> {
public:
  // types
  typedef boost::type_erasure::any< requirements, _iter > iterator_type;

  // construct/copy/destruct
  explicit decor_printer(std::size_t = 0, std::size_t = 0,
                         const std::string & = "\n",
                         const std::string & = " ",
                         const std::string & = "\n",
                         const std::string & = "\n");

  // public member functions
  template<typename Range, typename CharT, typename Traits>
    void print(const Range &,
               std::basic_ostream< CharT, Traits > & = std::cout) const;

  // protected member functions
  virtual void do_print(iterator_type, iterator_type, ostream_type) const;
};
```

Description

Outputs items in a specified number of columns across rows with a separator (often comma and/or space(s)), and a suffix (usually newline) every `num_columns` items.
Usage:

```
decor_printer simple_printer(3, 0, "\n", ", ", "\n", "\n");
```

Provide a container to be printed:

```
double da[] = {1., 2., 3., 4., 5., 6.};
```

Print to std::cout using:

```
simple_printer.print(std::cout, da);
```

Output:
1, 2, 3,
4, 5, 6,
7, 8, 9,
10, 11, 12
The order of parameters is chosen to try to allow use of the defaults as much as possible, including all defaults placing all items on one line or row separated by spaces.

`decor_printer` public construct/copy/destruct

1.
```
explicit decor_printer(std::size_t num_columns = 0, std::size_t wid = 0,
                       const std::string & pre = "\n",
                       const std::string & sep = " ",
                       const std::string & suf = "\n",
                       const std::string & term = "\n");
```

Constructor.
Usage: for example:

```
3, 10, double testd[] = {\n    ", ", ", "\n    ", "\n  };
```

3 columns with width 10, prefix newline, separator string comma space, suffix (newline at the end of a column), and a terminator string "
Defaults are provided for all parameters, so can simply construct:

```
my_default_printer decor_printer;
```

that places all items on one line or row with space between items, and a final newline.

`decor_printer` public member functions

1.
```
template<typename Range, typename CharT, typename Traits>
  void print(const Range & r,
             std::basic_ostream< CharT, Traits > & os = std::cout) const;
```

Parameters:	`os`	`std::ostream` for printing, default, for example: std::cout.
	`r`	Container to be printed, for example, a C array, std::vector, std::list ...
Template Parameters:	`Range`	must be a Forward Range whose elements can be printed to `std::ostream` os.

`decor_printer` protected member functions

1.
```
virtual void
do_print(iterator_type first, iterator_type last, ostream_type os) const;
```

Declare the pure virtual function `do_print` that is defined in all real printers.

Struct template concept_interface<base_and_derived< T, U >, Base, U>

boost::type_erasure::concept_interface<base_and_derived< T, U >, Base, U>

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename T, typename U, typename Base>
struct concept_interface<base_and_derived< T, U >, Base, U> : public Base {

  // public member functions
  operator typename rebind_any< Base, const T & >::type() const;
  operator typename rebind_any< Base, T & >::type();
};
```

Description

`concept_interface` public member functions

1.
```
operator typename rebind_any< Base, const T & >::type() const;
```

2.
```
operator typename rebind_any< Base, T & >::type();
```

Struct template ostreamable<Os, std::pair< T1, T2 >>

boost::type_erasure::ostreamable<Os, std::pair< T1, T2 >>

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename Os, typename T1, typename T2>
struct ostreamable<Os, std::pair< T1, T2 >> {

  // public static functions
  static void apply(Os &, const std::pair< T1, T2 > &);
};
```

Description

`ostreamable` public static functions

1.
```
static void apply(Os & out, const std::pair< T1, T2 > & arg);
```

Function template operator<<

boost::type_erasure::operator<<

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>


template<typename T1, typename T2>
  std::ostream & operator<<(std::ostream & os, const std::pair< T1, T2 > & p);
```

Description

Output a pair of values with space as delimiter.

For example: "1.23 3.45".

Header <boost/quan/unc.hpp>

Class for simple Propagation of Uncertainties according to a pure Gaussian model.

Based on code by Evan Manning (manning@alumni.caltech.edu) Evan Marshal Manning, C/C++ Users Journal, March 1996 page 29 to 38. original downloaded from ftp://beowulf.jpl.nasa.gov/pub/manning. This is a simple model of uncertainties, designed to accompany an article published in C/C++ Users Journal March 1996 at http://www.ddj.com/cpp/184403147 . A fuller collection of even fancier classes also given in Evan Marshal Manning's unc.h.

Standard deviation & its uncertainty added Paul A Bristow 31 Mar 98 ... .

Uncertainty as Applied to Measurements and Calculations John Denker http://www.av8n.com/physics/uncertainty.htm#sec-crank3 Here are some simple rules that apply whenever you are writing down a number:

1. Use many enough digits to avoid unintended loss of information.

2. Use few enough digits to be reasonably convenient.

3. For other calculations, keep a potentially significant digits (std::numeric_limits<>::max_digits10), 17 for double.

Possible Doxygen main page for uncertain classes, but will clash with svg_plot and rounding main pages.

Boost.Quan

This is the standalone Doxygen main page of the proposed Boost.Quan library.

This library provides a collection of classes and functions for handling uncertain reals.

Uncertainty as standard deviation and degrees of freedom are propagated through calculations using the uncertain type, and are input and output including the uncertainty estimates.

See Boost.Quan HTML Manual at

quan/libs/quan/doc/html/index.html

and/or equivalent PDF Manual at:

quan/libs/quan/doc/quan.pdf

Examples are in folder:

/boost/libs/quan/example/

and C++ include files are in folder:

/boost/boost/quan/

```
namespace boost {
  namespace quan {
    template<typename Type> struct lessAbs;

    class resetMaskedUncFlags;
    class resetUncFlags;
    class setAllUncFlags;
    class setConfidence;
    class setMaskedUncFlags;
    class setRoundingLoss;
    class setScale;
    class setSigDigits;
    class setUncFlags;
    class setUncSigDigits;
    class setUncWidth;
    class showUncFlags;
    class showUncTypes;
    template<bool is_correlated = false> class unc;

   // 16 type bits used by unc class uncTypes. Bit set = 1 means a positive attribute. Unctypes ↵
are
    enum unc_types { VALUE_ZERO = = 1 << 0, VALUE_INTEGER = = 1 << 1,
                     VALUE_RATIONAL = = 1 << 2,
                     VALUE_NEGATIVE_ONLY = = 1 << 3,
                     VALUE_POSITIVE_ONLY = = 1 << 4, UNC_KNOWN = = 1 << 5,
                     UNC_NOPLUS = = 1 << 6, UNC_NOMINUS = = 1 << 7,
                     UNC_QUAN_DECIMAL = = 1 << 8, UNC_QUAN_BINARY = = 1 << 9,
                     UNC_EXPLICIT = = 1 << 10, UNC_UNIFORM = = 1 << 11,
                     UNC_TRIANGULAR = = 1 << 12, DEG_FREE_EXACT = = 1 << 13,
                     DEG_FREE_KNOWN = = 1 << 14, SPARE = = 1 << 15 };
    enum uncIOflags;

    typedef unc< true > unccorr;  // Uncertainties are NOT correlated. Uncorrelated is the nor↵
mal case when uncertainties add.
    typedef unc< false > uncun;

    const unsigned short int DEG_FREE_DEF;
    static const long indexID;    // https://en.cppreference.com/w/cpp/language/operators oper↵
ator<< must be non-member functions, friend std::ostream& operator<<(std::ostream& os, const T& ↵
obj) https://en.cppreference.com/w/cpp/language/friend and may need to be friend to access ↵
private data of the class
    const unsigned short int UNC_DEF;
   const char * uncTypeWords;    // Description as a word of each bit in unc_type, using enum ↵
unc_types.[br].
    const unsigned short int VALUE_EXACT;    // Both set.
    std::ios_base & adddegfree(std::ios_base &);
    std::ios_base & addlimits(std::ios_base &);
    std::ios_base & addnoisyDigit(std::ios_base &);
    std::ios_base & addsiprefix(std::ios_base &);
    std::ios_base & addsisymbol(std::ios_base &);
    autoprefix_norm_impl< unc< false >, true >::type
    autoprefix_norm(const unc< false > & arg);
    std::ios_base & autoscale(std::ios_base &);
    std::ios_base & autosigdigits(std::ios_base &);
    std::ios_base & autouncsigdigits(std::ios_base &);
    std::ios_base & firmform(std::ios_base &);
    std::ios_base & flexform(std::ios_base &);
    std::ios_base & noautoscale(std::ios_base &);
    std::ios_base & nodegfree(std::ios_base &);
    std::ios_base & nolimits(std::ios_base &);
    std::ios_base & nonoisyDigit(std::ios_base &);
    std::ios_base & noplusminus(std::ios_base &);
    std::ios_base & noscale(std::ios_base &);
    std::ios_base & nosiprefix(std::ios_base &);
    std::ios_base & nosisymbol(std::ios_base &);
    std::ostream & operator<<(std::ostream &, const resetMaskedUncFlags &);
    std::ostream & operator<<(std::ostream & os, const resetUncFlags & sf);
    std::ostream & operator<<(std::ostream & os, const setAllUncFlags & sf);
    std::ostream & operator<<(std::ostream &, const setConfidence &);
    std::ostream & operator<<(std::ostream & os, const setMaskedUncFlags & sf);
    std::ostream & operator<<(std::ostream &, const setRoundingLoss &);
    std::ostream & operator<<(std::ostream & os, const setScale & sc);
    std::ostream & operator<<(std::ostream & os, const setSigDigits & sf);
    std::ostream & operator<<(std::ostream & os, const setUncFlags & sf);
    std::ostream & operator<<(std::ostream & os, const setUncSigDigits & usf);
    std::ostream & operator<<(std::ostream & os, const setUncWidth & sw);

    // Output uncFlags as descriptive word strings to this std::ostream.
    std::ostream & operator<<(std::ostream & os, const showUncFlags & uf);
    std::ostream & operator<<(std::ostream &, const showUncTypes &);
    template<bool correlated>
      std::ostream &
      operator<<(std::ostream &,
                 const std::pair< unc< correlated >, unc< correlated > > &);
    std::ostream & operator<<(std::ostream &, const unc< false > &);
    std::istream &
    operator>>(std::istream & is, const resetMaskedUncFlags & sf);
    std::istream & operator>>(std::istream & is, const resetUncFlags & sf);
    std::istream & operator>>(std::istream & is, const setAllUncFlags & sf);
    std::istream & operator>>(std::istream & is, const setMaskedUncFlags & sf);
    std::istream & operator>>(std::istream & is, const setScale & sc);
    std::istream & operator>>(std::istream & is, const setSigDigits & sf);
    std::istream & operator>>(std::istream & is, const setUncFlags & sf);
    std::istream & operator>>(std::istream & is, const setUncSigDigits & usf);
    std::istream & operator>>(std::istream & is, const setUncWidth & sw);

    // Extract operator<< for unc type.
    std::istream & operator>>(std::istream & is, unc< false > & ud);
    void outUncIOFlags(long, std::ostream & = std::cerr, std::string = ".\n");
    void outUncIOFlags(std::ostream & = std::cout, std::ostream & = std::cerr,
                       std::string = ".\n");
    void outUncTypes(unsigned short int, std::ostream & = std::cerr);
    void outUncValues(std::ostream & = std::cout, std::ostream & = std::cerr);
    std::ios_base & plusminus(std::ios_base &);

    // Quaded function, notationally convenient for x^4, but overflow possible? Used by Welch-
Satterthwaite formula.
    template<typename Type> Type pow4(const Type & a);
    long resetuFlags(std::ios_base & str, long flags);
    std::ios_base & scale(std::ios_base &);
    std::ios_base & setsigdigits(std::ios_base &);
    long setuFlags(std::ios_base & str, long flags);
    void setUncDefaults(std::ios_base &);
    std::ios_base & setuncsigdigits(std::ios_base &);
    std::ios_base & showuncflags(std::ios_base & iostr);
    long uFlags(std::ios_base & str);
    long uFlags(std::ios_base & str, long flags);
    void unc_input(double &, double &, unsigned short int &,
                   unsigned short int &, std::istream &);
    template<> double unc_of(double);
    template<> double unc_of(float);
    template<typename T> double unc_of(T);
    template<> double unc_of(uncun);
    template<typename T>
      const std::pair< float, float > uncs_of(std::pair< const T, T >);
    template<>
```

```
      std::pair< double, double > uncs_of(std::pair< double, double >);
    template<typename T>
      std::pair< double, double > uncs_of(std::pair< T, T >);
    template<typename T1, typename T2>
      std::pair< double, double > uncs_of(std::pair< T1, T2 >);
    template<> std::pair< double, double > uncs_of(std::pair< uncun, uncun >);
    template<> double value_of(double);
    template<typename T> double value_of(T);
    template<> double value_of(unc< false >);
    template<> double value_of(unc< true >);
    template<>
      std::pair< double, double > values_of(std::pair< double, double >);
    template<typename T>
      std::pair< double, double > values_of(std::pair< T, T >);
    template<typename T1, typename T2>
      std::pair< double, double > values_of(std::pair< T1, T2 >);
    template<>
      std::pair< double, double > values_of(std::pair< uncun, uncun >);
  }
}
 BOOST_STATIC_ASSERT(std::numeric_limits< double >::is_iec559);
```

Struct template lessAbs

boost::quan::lessAbs — Predicate comparison operators for use by sort etc. Functors are *preferred* to functions for STL algorithms: Scott Meyers, ESTL, item 46, page 201...

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<typename Type>
struct lessAbs : public std::function< Type > {

  // public member functions
  bool operator()(const Type &, const Type &) const;
};
```

Description

Used below by min_element, etc.

`lessAbs` public member functions

1.
```
bool operator()(const Type & a, const Type & b) const;
```

Class resetMaskedUncFlags

boost::quan::resetMaskedUncFlags

Synopsis

```
// In header: <boost/quan/unc.hpp>


class resetMaskedUncFlags {
public:
  // construct/copy/destruct
  resetMaskedUncFlags(int, int);

  // friend functions
  std::ostream & operator<<(std::ostream &, const resetMaskedUncFlags &);

  // public data members
  int flags_;
  int mask_;
};
```

Description

`resetMaskedUncFlags` public construct/copy/destruct

1.
```
resetMaskedUncFlags(int w, int m);
```

`resetMaskedUncFlags` friend functions

1.
```
std::ostream & operator<<(std::ostream &, const resetMaskedUncFlags &);
```

   Reset (clear) the bits specified in the `uncFlagsIndex`.

   Example Usage:

```
std::cout << setMaskedUncFlags(0xF) ...
```

Class resetUncFlags

boost::quan::resetUncFlags — Reset (clear = 0) selected uncertain class flags for a std::stream.

Synopsis

```
// In header: <boost/quan/unc.hpp>


class resetUncFlags {
public:
  // construct/copy/destruct
  resetUncFlags(int);

  // public data members
  int flags_;
};
```

Description

Usage:

```
std::cout << resetUncFlags(0x7) ...
```

See Also:

`setUncFlags` to set uncertain class flags.

`resetUncFlags` public construct/copy/destruct

1.
   ```
   resetUncFlags(int w);
   ```

Class setAllUncFlags

boost::quan::setAllUncFlags — setAllUncflags(int flags);

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setAllUncFlags {
public:
  // construct/copy/destruct
  setAllUncFlags(int);

  // public data members
  int flags_;
};
```

Description

Example Usage:

```
std::cout << setAllUncFlags(0x5a) ...
```

`setAllUncFlags` public construct/copy/destruct

1.
   ```
   setAllUncFlags(int flags);
   ```

Class setConfidence

boost::quan::setConfidence — setConfidence(int setConfidence); to control the confidence interval. Usage: `out << setConfidence(0.01)` ...

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setConfidence {
public:
  // construct/copy/destruct
  setConfidence(double);

  // public data members
  double confidence_;
};
```

Description

`setConfidence` public construct/copy/destruct

1.
```
setConfidence(double);
```

   setConfidence(double alpha); Usage: out << `setConfidence`(0.01) ... = cout.iword[confidence] = 0.01;

Class setMaskedUncFlags

boost::quan::setMaskedUncFlags — setMaskedUncflags(int flags, int mask);

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setMaskedUncFlags {
public:
  // construct/copy/destruct
  setMaskedUncFlags(int, int);

  // public data members
  int flags_;
  int mask_;
};
```

Description

Example:

```
std::cout << setMaskedUncFlags(0x7, 0x3) ...
```

See Also:

setUncFlags if no mask is required.

`setMaskedUncFlags` public construct/copy/destruct

1.
```
setMaskedUncFlags(int w, int m);
```

Class setRoundingLoss

boost::quan::setRoundingLoss — setRoundingLoss(int setRoundingLoss); to control the acceptable rounding loss. Usage: `out << setRoundingLoss(0.01)` ...

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setRoundingLoss {
public:
  // construct/copy/destruct
  setRoundingLoss(double);

  // public data members
  double roundingloss_;
};
```

Description

`setRoundingLoss` public construct/copy/destruct

1.
   ```
   setRoundingLoss(double);
   ```

   setroundingLoss(double eps); Usage: out << setroundingLoss(0.01) ... = cout.iword[roundingLoss] = 0.01;

Class setScale

boost::quan::setScale

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setScale {
public:
  // construct/copy/destruct
  setScale(int);

  // public data members
  int scale;
};
```

Description

`setScale` public construct/copy/destruct

1.
   ```
   setScale(int);
   ```

Class setSigDigits

boost::quan::setSigDigits

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setSigDigits {
public:
  // construct/copy/destruct
  setSigDigits(int);

  // public data members
  int sigDigits_;
};
```

Description

`setSigDigits` public construct/copy/destruct

1.
   ```
   setSigDigits(int);
   ```

   Set the number of significant digits that can be used. (If required by `<< sigfiged` then use set sigDigits value, or noSigfiged to **NOT** use sigDigits). Usage: `out << setSigDigits(5) << setsigdigits ...` & either `<< setsigdigits` to use set sigDigits value, or `<< nosetsigdigits` to not use the set sigDigits.

   setSigDigits(int sigDigits);

Class setUncFlags

boost::quan::setUncFlags — Set selected uncertain flags.

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setUncFlags {
public:
  // construct/copy/destruct
  setUncFlags(int);

  // public data members
  int flags_;
};
```

Description

Usage:

```
out << setUncFlags(0x7) ...
```

`setUncFlags` public construct/copy/destruct

1.
   ```
   setUncFlags(int flags);
   ```

Class setUncSigDigits

boost::quan::setUncSigDigits

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setUncSigDigits {
public:
  // construct/copy/destruct
  setUncSigDigits(int);

  // public data members
  int uncSigDigits_;
};
```

Description

setUncSigDigits(int uncSigDigits); Permits choice of number of uncertain or stddev value:
2 is the ISO recommendation.
Uncertainty of measurement – Part 3: Guide to the expression of uncertainty in measurement (GUM:1995)
ISO Guide 98 (1995) and updated version 2008.
1 is more appropriate for very small degrees of freedom, (but it gives a big step when the value starts with 1 or 2, when the difference between 1.4 (rounded to 1) and 1.6 (rounded to 2.) is a doubling. 3 is appropriate only for large degrees of freedoms, >= 1000.

Warning

Values < 1 or > 3 are silently ignored. -1 passes through to allow dynamic choice of uncertainty significant digits or precision based on degrees of freedom. Usage: out << setUncSigDigits(3) ...

`setUncSigDigits` public construct/copy/destruct

1.
   ```
   setUncSigDigits(int);
   ```

   setUncSigDigits(int sigDigits); Usage: `out << setUncSigDigits(3) ...`

Class setUncWidth

boost::quan::setUncWidth

Synopsis

```
// In header: <boost/quan/unc.hpp>


class setUncWidth {
public:
  // construct/copy/destruct
  setUncWidth(int);

  // public data members
  int uncWidth;
};
```

Description

`setUncWidth` public construct/copy/destruct

1.
```
setUncWidth(int);
```

Class showUncFlags

boost::quan::showUncFlags

Synopsis

```
// In header: <boost/quan/unc.hpp>


class showUncFlags {
public:
  // construct/copy/destruct
  showUncFlags(unsigned short int);

  // friend functions
  std::ostream & operator<<(std::ostream &, const showUncFlags &);

  // public data members
  unsigned short int flags;
};
```

Description

`showUncFlags` public construct/copy/destruct

1.
```
showUncFlags(unsigned short int);
```

Constructor. Usage: out << `showUncFlags`(static_cast<long>(is.iword(uncFlagsIndex))) ...

`showUncFlags` friend functions

1.
```
std::ostream & operator<<(std::ostream &, const showUncFlags &);
```

Output uncFlags as descriptive word strings to this `std::ostream`.

Class showUncTypes

boost::quan::showUncTypes — Helper function for `showUncTypes` friend `std::ostream` operator<<.

Synopsis

```
// In header: <boost/quan/unc.hpp>


class showUncTypes {
public:
  // construct/copy/destruct
  showUncTypes(unsigned short int);

  // friend functions
  std::ostream & operator<<(std::ostream &, const showUncTypes &);

  // public data members
  unsigned short int types_;
};
```

Description

`showUncTypes` public construct/copy/destruct

1.
```
showUncTypes(unsigned short int t);
```

`showUncTypes` friend functions

1.
```
std::ostream & operator<<(std::ostream &, const showUncTypes &);
```

   `Extract` operator<< to show all the 16 types of an uncertain item. Example:

```
std::cout << showUncTypes(uncType)
```

Class template unc

boost::quan::unc

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<bool is_correlated = false>
class unc : public std::char_traits< char > {
public:
  // types
  typedef double value_type;

  // construct/copy/destruct
  unc(const double, const float = 0.0f, const short unsigned int = 0,
      const short unsigned int = UNC_KNOWN|UNC_EXPLICIT|DEG_FREE_EXACT|DEG_FREE_KNOWN);
  explicit unc(const int = 0, const float = 0.0f,
               const short unsigned int = 0,
               const short unsigned int = DEG_FREE_DEF|VALUE_INTEGER|VALUE_RATION↵
AL|UNC_NOPLUS|UNC_NOMINUS|UNC_KNOWN);
  unc(const unc &);

  // friend functions
  void unc_input(double &, double &, unsigned short int &,
                 unsigned short int &, std::istream &);

  // public member functions
  unsigned short int deg_free() const;
  void deg_free(short unsigned int);
  unsigned short int degFree(void) const;
  float deviation(void) const;
  double mean(void) const;
  operator value_type() const;
  bool operator!=(const unc< is_correlated > &) const;
  unc< is_correlated > & operator*=(const double &);
  unc< is_correlated > & operator*=(const unc< is_correlated > &);
  unc< is_correlated > operator+(void) const;
  unc< is_correlated > operator++(int);
  unc< is_correlated > operator++(void);
  unc< is_correlated > & operator+=(const double &);
  unc< is_correlated > & operator+=(const unc< is_correlated > &);
  unc< is_correlated > operator-(void) const;
  unc< is_correlated > operator--(int);
  unc< is_correlated > operator--(void);
  unc< is_correlated > & operator-=(const double &);
  unc< is_correlated > & operator-=(const unc< is_correlated > &);
  unc< is_correlated > & operator/=(const double &);
  unc< is_correlated > & operator/=(const unc< is_correlated > &);
  bool operator<(const unc< is_correlated > &) const;
  bool operator==(const unc< is_correlated > &) const;
  void setUncTypes(short unsigned int);
  float std_dev() const;
  void std_dev(float);
  unsigned short int types() const;
  void types(short unsigned int);
  unsigned short int uncFlags(void) const;
  double value() const;
  void value(double);

  // public static functions
  static bool equalU(const unc< is_correlated > &,
                     const unc< is_correlated > &);
  static bool equalU2(const unc< is_correlated > &,
                      const unc< is_correlated > &);
  static bool lessU(const unc< is_correlated > &,
                    const unc< is_correlated > &);
  static bool lessU2(const unc< is_correlated > &,
                     const unc< is_correlated > &);
  static bool moreU(const unc< is_correlated > &,
                    const unc< is_correlated > &);

  // public data members
  short unsigned int degFree_;
  float uncertainty_;  // Measure of uncertainty, typically, standard deviation, if known.
  short unsigned int unctypes_;  // Information about the value and uncertainty, encoded as a ↲
bitmap 16 bits, 0 to 15.
  double value_;  // aka mean, estimate, central or most likely value.
};
```

Description

Uncertain number template class unc, using mean and measure of uncertainty (perhaps standard deviation if pure Gaussian distribution), but also includes information about uncertainty as degrees of freedom & distribution (default Gaussian).

Template Parameters

1.
```
bool is_correlated = false
```

if true, standard deviation is correlated, else not correlated (the common case).

unc public construct/copy/destruct

1.
```
unc(const double val, const float unc = 0.0f, const short unsigned int df = 0,
    const short unsigned int uncTypeFlags = UNC_KNOWN|UNC_EXPLICIT|DEG_FREE_EX↲
ACT|DEG_FREE_KNOWN);
```

Default constructor from double value & float uncertainty. Constructor from just a double value assumed exact, so behaves like a normal double. Normal conversion from double to unc. (See also constructor from integer which alone sets integer flag.)

2.
```
explicit unc(const int ivalue = 0, const float unc = 0.0f,
             const short unsigned int df = 0,
             const short unsigned int uncTypeFlags = DEG_FREE_DEF|VALUE_INTEGER|VALUE_RATION↲
AL|UNC_NOPLUS|UNC_NOMINUS|UNC_KNOWN);
```

< unc Constructor from integer value.

3.
```
unc(const unc & ud);
```

unc friend functions

1.
```
void unc_input(double & mean, double & stdDev,
               unsigned short int & degreesOfFreedom,
               unsigned short int & types, std::istream & is);
```

Uncertainties ARE correlated. This is the unusual case where the sum must be exact, so the uncertainties are subtracted. Example: two blocks which fit into a box perfectly. So if both have an uncertainties, they must cancel when the uncertainties are added. Also applies to items like concentrations which must add up to 100% or unity.

Inputs uncertainty as value, (implicitly exact, std deviation = 0). & optionally an explicit measure of uncertainty [[+]|[-] <standard deviation * 2. >], (1.0 implies 1. +|- 0.5 and sd of 0.5, 1.00 implies 1. +|- 0.05 and sd of 0.05)

& optionally degrees of freedom [(<short int>)] like (99) Used by istream& operator>> (istream&, unc<is_correlated>&) Original simple version: char plus, slash, minus; s >> value >> plus >> slash >> minus >> stdDev; if ((plus != '+') || (slash != '/') || (minus != '-')) { cerr << "Unexpected characters encountered in reading " "value +/- stdDev !" << endl; is.setf(ios_base::failbit);

Used by std::istream& operator>> (std::istream& is, const unc }

unc public member functions

1.
```
unsigned short int deg_free() const;
```

Returns: Estimate of number of degrees of freedom, usually = number of observations -1.

2.
```
void deg_free(short unsigned int df);
```

Parameters: `df` Number ofdegrees of freedom, usually = number of observations -1, so 0 = means just one observation.

3.
```
unsigned short int degFree(void) const;
```

4.
```
float deviation(void) const;
```

5.
```
double mean(void) const;
```

6.
```
operator value_type() const;
```

7.
```
bool operator!=(const unc< is_correlated > & x) const;
```

8.
```
unc< is_correlated > & operator*=(const double & a);
```

9.
```
unc< is_correlated > & operator*=(const unc< is_correlated > & ud);
```

10.
```
unc< is_correlated > operator+(void) const;
```

11.
```
unc< is_correlated > operator++(int);
```

12.
```
unc< is_correlated > operator++(void);
```

13.
```
unc< is_correlated > & operator+=(const double & a);
```

14.
```
unc< is_correlated > & operator+=(const unc< is_correlated > & ud);
```

15.
```
unc< is_correlated > operator-(void) const;
```

16.
```
unc< is_correlated > operator--(int);
```

17.
```
unc< is_correlated > operator--(void);
```

18.
```
unc< is_correlated > & operator-=(const double & a);
```

19.
```
unc< is_correlated > & operator-=(const unc< is_correlated > & ud);
```

20.
```
unc< is_correlated > & operator/=(const double & a);
```

operator /=

Note

Dividing by constant doesn't change degrees of freedom or uncertainty. All indicators & restrictions on value removed. Must assume double is irrational and not integer, NOT int/int becomes rational as operator /= .

21.
```
unc< is_correlated > & operator/=(const unc< is_correlated > & ud);
```

22.
```
bool operator<(const unc< is_correlated > & x) const;
```

Predicate compare operator< for use by std::sort etc. (Note const needed to use with less!)

23.
```
bool operator==(const unc< is_correlated > & x) const;
```

24.
```
void setUncTypes(short unsigned int type);
```

Parameters:	`type`	bits indicating type of uncertain real.

25.
```
float std_dev() const;
```

Get estimate of uncertainty of value of uncertain type.

Returns: Estimate of uncertainty as standard deviation of value of uncertain type.

26.
```
void std_dev(float unc);
```

Set estimate of uncertainty of value of uncertain type.

Parameters:	`unc`	Estimate of uncertainty as standard deviation of value of uncertain type.

27.
```
unsigned short int types() const;
```

Returns: Types of uncertain real, encoded as a bitmap.

28.
```
void types(short unsigned int type);
```

29.
```
unsigned short int uncFlags(void) const;
```

30.
```
double value() const;
```

Get central estimate of value of uncertain type.

Returns: Central 'best' estimate of value of uncertain type.

31.
```
void value(double value);
```

Set central estimate of value of uncertain type.

Parameters: `value` Central estimate of value of uncertain type.

unc public static functions

1.
```
static bool equalU(const unc< is_correlated > & l,
                   const unc< is_correlated > & r);
```

2.
```
static bool equalU2(const unc< is_correlated > & l,
                    const unc< is_correlated > & r);
```

3.
```
static bool lessU(const unc< is_correlated > & l,
                  const unc< is_correlated > & r);
```

Compare two uncertain values.

Parameters:	`l`	Uncertain value.
	`r`	Uncertain value.
Returns:	true if l is effectively less than r.	

4.
```
static bool lessU2(const unc< is_correlated > & l,
                   const unc< is_correlated > & r);
```

5.
```
static bool moreU(const unc< is_correlated > & l,
                  const unc< is_correlated > & r);
```

unc public public data members

1.
```
short unsigned int degFree_;
```

Degrees of freedom, usually = number of observations -1, so zero if just one observation or measurement. Range from 0 (usually 1 observation) to 65534 = std::numeric_limits<unsigned short int>::max() - 1 so for 2 observations assign 1 to degFree_ degree of freedom,for 3 observations assign 2 .... Higher numbers of observations are indistinguishable from infinite observations. Max unsigned value 0xFFFF == 65535 is used to indicate degFree_ is NOT meaningful. BUT many programs seem to use NON-integer degrees of freedom, so a float might seem better, but fundamental `float` is 32 bits not 16, so use 16 floating-point type for compact struct?

2.
```
float uncertainty_;
```

Note

Reduced precision (float guarantees 6 decimal digits not 15) and range e38 not E304 should not be a problem unless value is (near) less than 1e38. Can be zero, meaning exact, and can be negative or anti-correlated, for example when values must add up to a total like 100%. Relative (Fraction) Coefficent of variation = Standard deviation / value (aka % +|- /100) Relative is a problem if value near zero. +|- is std deviation, so for input of "1.0" implicit standard deviation and uncertainty limits +|- 0.05.

3.
```
short unsigned int unctypes_;
```

See Also:

enum unc_types.

Note

These sizes mean that total size of a unc is 64 = 32 + 32 = 128 bits for IEEE-754 systems with 64-bit double.

4.
```
double value_;
```

Note

It is convenient to use 64-bit floating-point format for mean of central value (so even really accurate values like weights can be precise enough), 32-bit floating-point is ample accuracy for standard deviation fractional variation, leaving two 16-bit for degrees of freedom and other flags, so that total is same as two doubles & can be efficiently aligned.

Type uncIOflags

boost::quan::uncIOflags — Control of printing uncertain values, similar to `std::ios` flags.

Synopsis

```
// In header: <boost/quan/unc.hpp>


enum uncIOflags { defaults = = 0, firm = = 1  << 0, setScaled = = 1 << 1, 
                  autoScaled = = 1 << 2, plusMinus = = 1 << 3, 
                  addSISymbol = = 1 << 4, addSIPrefix = = 1 << 5, 
                  noisyDigit = = 1 << 6, useSetSigDigits = = 1 << 7, 
                  useSetUncSigDigits = = 1 << 8, degfree = = 1 << 9, 
                  replicates = = 1 << 0xA, limits = = 1 << 0xB };
```

Description

Can be output for diagnostic use, for example:

```
outUncIOFlags(std::cout.flags(), std::cerr); // uncFlags (0x201) firm adddegfree.
```

`defaults`	Default.
`firm`	bit 0: == 0 == false means flexible layout, or firm == 1 means true.
`setScaled`	bit 1 Set scaled == 1 or not scaled == 0. by fixed factor SetScale and << scale ... power10 in range -max to +max.
`autoScaled`	bit 2: auto = 1 with << autoscale ...
`plusMinus`	bit 3 = 1 means add +/- uncertainty Usage: `std::cout << plusminus << u;`
`addSISymbol`	bit 4 = 1, add suffix SI symbol G, M, k, m, u, n, p ... If one is applicable, else = 0 do nothing.
`addSIPrefix`	bit 5 = 1, add suffix SI prefix Giga, Mega, kilo ...
`noisyDigit`	Add extra 'noisy' decimal digit to value & sd. Usage: std::cout `<< addnoisyDigit << u;` This means is suitable for input to other calculations because the random extra 2 to 3 bits will approximate a continuous function. Quantization to fewer digits would distort statistical calculations. Default is to use minimum decimal digits for clearer display. Extra digit also added when degrees of freedom > 10.
`useSetSigDigits`	if bit 7 = 1 use set sig digits else calculate from sd(isUseSetSigDigits).
`useSetUncSigDigits`	if bit 8 = 1 use set Unc sigfig (isStdDevSigDigitsSet). else calculate from degrees of freedom.
`degfree`	0x200, If bit 9 == 1, add output of degrees of freedom as (99).
`replicates`	0x400 If == 1, add output of replicates as [99] if > 1.
`limits`	0X800 add limits.

Global DEG_FREE_DEF

boost::quan::DEG_FREE_DEF

Synopsis

```
// In header: <boost/quan/unc.hpp>

const unsigned short int DEG_FREE_DEF;
```

Global indexID

boost::quan::indexID — https://en.cppreference.com/w/cpp/language/operators operator<< must be non-member functions, friend std::ostream& operator<<(std::ostream& os, const T& obj) https://en.cppreference.com/w/cpp/language/friend and may need to be friend to access private data of the class

Synopsis

```
// In header: <boost/quan/unc.hpp>

static const long indexID;
```

Description

'Unique' value used to check xalloc initialised iwords are OK, and are not corrupted.

Global UNC_DEF

boost::quan::UNC_DEF

Synopsis

```
// In header: <boost/quan/unc.hpp>

const unsigned short int UNC_DEF;
```

Global uncTypeWords

boost::quan::uncTypeWords — Description as a word of each bit in `unc_type`, using enum unc_types.[br].

Synopsis

```
// In header: <boost/quan/unc.hpp>

const char * uncTypeWords;
```

Description

These bits record the type of value stored, for example: VALUE_ZERO, UNC_KNOWN, DEG_FREE_KNOWN, UNC_UNIFORM... Used by function outUncTypes. Example:

Global VALUE_EXACT

boost::quan::VALUE_EXACT — Both set.

Synopsis

```
// In header: <boost/quan/unc.hpp>

const unsigned short int VALUE_EXACT;
```

Function adddegfree

boost::quan::adddegfree — Set `std::ostream` to add degrees of freedom to output of uncertain class `unc` as, for example: (99).

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & adddegfree(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << nodegreefree << u << std::endl;
```

Parameters:	`iostr` `std::ostream` for output of uncertain class `unc` value.
Returns:	`std::ostream` to make chainable.

Function addlimits

boost::quan::addlimits — Set stream to add confidence limits as interval for example: <1.23, 1.26>.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & addlimits(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << addlimits << u << std::endl;
```

Parameters:	`iostr` `std::ostream` for confidence limits.
Returns:	`std::ostream` to make chainable.

Function addnoisyDigit

boost::quan::addnoisyDigit — Set `std::ostream` so that an extra probably 'noisy' digit is added when for output of **value** (mean) of class `unc`. This is useful if the value is being read into some other system thus avoiding digital quantization errors.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & addnoisyDigit(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << addnoisyDigit << u ...
```

Parameters:	`iostr` `std::ostream` for which values should add a noisy digit.
Returns:	`std::ostream` to make chainable.

Function addsiprefix

boost::quan::addsiprefix — Set `std::ostream` so that an appropriate SI prefix is added when for output of **value** (mean) of class `unc`. This is NOT the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & addsiprefix(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << addsiprefix << u ...
```

Parameters:	`iostr` `std::ostream` for which values should add an SI prefix.
Returns:	`std::ostream` to make chainable.

Function addsisymbol

boost::quan::addsisymbol — Set `std::ostream` so that an appropriate SI symbol is added when for output of **value** (mean) of class `unc`. This is NOT the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & addsisymbol(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << addsisymbol << u ...
```

Parameters:	`iostr` `std::ostream` for which values should add an SI prefix.
Returns:	`std::ostream` to make chainable.

Function autoscale

boost::quan::autoscale — Set `std::ostream` so that autoscaling of output values takes place for output of **value** (mean) of class `unc`. This is NOT the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & autoscale(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << autoscale << u ...
```

< set bit 2 = 1 to mean auto.

Parameters:	`iostr` `std::ostream` for which values should be autoscaled.
Returns:	`std::ostream` to make chainable. Scaling can also be applied by a preset factor using manipulator

```
<< scale <<
```

.

Function autosigdigits

boost::quan::autosigdigits — Set `std::ostream` so that the number of significant digits (prcision) for output of **value** (mean) of class `unc` is computed from the uncertain class `unc` itself.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & autosigdigits(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << autosigdigits ...
```

will use precision computed from an estimate of the uncertainty (standard deviation) and degrees of freedom.

Use auto (calculated from uncertainty) not sig digits stored with `<< setSigDigits(6)` for value.

Parameters:	`iostr` `std::ostream` for which to set precision to use to show uncertainty.
Returns:	`std::ostream` to make chainable.

Function autouncsigdigits

boost::quan::autouncsigdigits — Set `std::ostream` so that the number of significant digits (prcision) for output of uncertainty (standard deviation) of class `unc` is computed from the uncertain class `unc` itself.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & autouncsigdigits(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
<< autouncsigdigits ...
```

will use precision computed from an estimate of the uncertainty of standard deviation and degrees of freedom.

Calculate stdDev sig digits from uncertainty.

Parameters:	`iostr`	`std::ostream` for which to set precision to show uncertainty.
Returns:	`std::ostream` to make chainable.	

Function firmform

boost::quan::firmform

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & firmform(std::ios_base & iostr);
```

Description

< set bit 0 = 1 to mean firm not flex.

Function flexform

boost::quan::flexform — Parameterless manipulators to switch format by changing uncFlag bits, flex - adjust width just enough to display, suitable for in-line display of non-tables, or firm to fit into a set width, suitable for tables.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & flexform(std::ios_base & iostr);
```

Description

Similarly for scale and autoscale flag bits which use either a preset scaling factor or autoscaled, or the default noscaling. Note names use all lowercase to match convention of std::hex, std::oct, std::showpos ...[br] Usage:

```
std::cout << scale << noscale << autoscale << noautoscale ...
```

< clear bit 0 = 0 to mean flex.

Function noautoscale

boost::quan::noautoscale — Set `std::ostream` so that no autoscaling of output values takes place for output of **value** (mean) of class `unc`. This is the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & noautoscale(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << noautoscale << u ...
```

< clear bit 2 = 0 to mean not autoscaled.

Parameters:	`iostr`	`std::ostream` for which values should not be autoscaled.
Returns:	`std::ostream` to make chainable.	

Function nodegfree

boost::quan::nodegfree — Set stream to NOT add degrees of freedom as for example: (99).

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & nodegfree(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << nodegreefree << u << std::endl;
```

Parameters:	`iostr`	`std::ostream` for confidence limits.
Returns:	`std::ostream` to make chainable.	

Function nolimits

boost::quan::nolimits — Set stream to NOT add confidence limits as interval for example: <1.23, 1.26>.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & nolimits(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << noaddlimits << u << std::endl;
```

Parameters:	`iostr`	`std::ostream` for confidence limits.
Returns:	`std::ostream` to make chainable.	

Function nonoisyDigit

boost::quan::nonoisyDigit — Set `std::ostream` so that an extra probably 'noisy' digit is NOT added when for output of **value** (mean) of class `unc`. (This is only useful if the value is being read into some other system thus avoiding digital quantization errors). No noisy digit is the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & nonoisyDigit(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << addnoisyDigit << u ...
```

Parameters:	`iostr`	`std::ostream` for which values should ad a noisy digit.
Returns:	`std::ostream` to make chainable.	

Function noplusminus

boost::quan::noplusminus — Set `std::ostream` so that NO uncertainty estimate, usually standard deviation, is added when for output of **value** (mean) of class `unc`. This is the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & noplusminus(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << noplusminus << u ...
```

outputs a mean value only.

Parameters:	`iostr`	`std::ostream` for which values should add an uncertainty estimate.
Returns:	`std::ostream` to make chainable.	

Function noscale

boost::quan::noscale — Set `std::ostream` so that NO scaling of output values to a preset factor takes place for output of **value** (mean) of class `unc`. This is the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & noscale(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << noscale << u ...
```

< clear bit 1 = 0 to mean not scaled.

Parameters:	`iostr`	`std::ostream` for which values should be scaled.
Returns:	`std::ostream` to make chainable.	

Function nosiprefix

boost::quan::nosiprefix — Set `std::ostream` so that no SI prefix is added when for output of **value** (mean) of class `unc`. This is the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & nosiprefix(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << addnoisyDigit << u ...
```

Parameters:	`iostr`	`std::ostream` for which values should add a noisy digit.
Returns:	`std::ostream` to make chainable.	

Function nosisymbol

boost::quan::nosisymbol — Set `std::ostream` so that no SI symbol is added when for output of **value** (mean) of class `unc`. This is the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & nosisymbol(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << nosisymbol << u ...
```

Parameters:	`iostr`	`std::ostream` for which values should add an SI prefix.
Returns:	`std::ostream` to make chainable.	

Function operator<<

boost::quan::operator<< — Reset (clear) the bits specified in the `uncFlagsIndex`.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ostream & operator<<(std::ostream & os, const resetMaskedUncFlags & sf);
```

Description

Example Usage:

```
std::cout << setMaskedUncFlags(0xF) ...
```

Function operator<<

boost::quan::operator<<

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ostream & operator<<(std::ostream & os, const setConfidence & sl);
```

Description

Note

Can't store `double` in a `long`, so scale up to an integer.

Function operator<<

boost::quan::operator<<

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ostream & operator<<(std::ostream & os, const setRoundingLoss & sl);
```

Description

Note

Can't store `double` in a `long`, so scale up to an integer.

Function operator<<

boost::quan::operator<< — `Extract` operator<< to show all the 16 types of an uncertain item. Example:

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ostream & operator<<(std::ostream & os, const showUncTypes & ut);
```

Description

```
std::cout << showUncTypes(uncType)
```

Parameters:		
	`os`	`std::ostream&` for output of uncertain types as descriptive words, for example: integer, zero, df_exact.
	`ut`	Uncertain type flags (bits) for the `std::ostream`.

Function template operator<<

boost::quan::operator<<

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<bool correlated>
  std::ostream &
  operator<<(std::ostream & os,
             const std::pair< unc< correlated >, unc< correlated > > & up);
```

Description

Output a pair (X and Y) of uncertain values with (if defined) uncertainty and degrees of freedom.

For example: "1.23 +/- 0.01 (13), 3.45 +/- 0.06 (78)".

Function operator<<

boost::quan::operator<< — Extract `operator<<` for uncertain types. Example:

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ostream & operator<<(std::ostream & os, const unc< false > & val);
```

Description

```
uncun u(1.23, 0.05, 9); std::cout << u << std::endl;
```

(Should cover both correlated and uncorrelated cases?)
boost::quan::unc::operator<<(std::ostream& os, const unc<false>& val) boost::quan::unc::operator<<(std::ostream& os, const unc<is_correlated>& val)

< Add an extra 'noisy' guard digit to reduce risk of information loss.

< Append degrees of freedom.

< Uncertainty as +/- is required too (but ignore if value is exact or integer).

Exponential format is, for example, 1E6 else 1e6.

Taken to mean that exponential format wanted (always possible).

< Means decimal point is always shown, for example 900. even if not needed.

< Show + sign always because `std::ios` flag was set with << showpoint.

<

```
os << fixed ...
```

ios decimal fixed d.dddd format (rather than scientific).

<

```
os << setw(9)
```

has prescribed a width (rather than default width == 0).

< = default but unc usage not defined yet, center?

< Right justify, prepend leading pad before.

```
<< right << ...
```

< Left justify, append trailing pad after.

```
<< left ...
```

< Not defined yet, but use to center in field?

< Center if BOTH left and right specified.

< Align on decimal point?

< Append confidence interval, for example, "<1.23, 1.56>"

< Use set sigdigits instead of calculate from uncertainty.

< Use setUNCsigdigits instead of calculate from uncertainty.

Warning

Width must be read BEFORE any use of os

Confidence or alpha to compute confidence interval is similarly scaled. Example:

```
std::cout << confidence(0.01) << ...
```

means 1 - confidence = 99% confidence.
The confidence is stored in a `long` integer scaled by 1000 thus:

```
double confidence = os.iword(conf) / 1000.;
```

Function outUncIOFlags

boost::quan::outUncIOFlags

Synopsis

```
// In header: <boost/quan/unc.hpp>


void outUncIOFlags(long uncFlags, std::ostream & os = std::cerr,
                   std::string terminator = ".\n");
```

Description

Show the set uncertain class io stream& flags settings as words, for examples: add_/-, set_scaled, addlimits.

See Also:

Version that takes the `uncflags` from `std::ostream&` as parameter.

```
void outUncIOFlags(std::ostream& os = std::cout, std::ostream& osout = std::cerr, std::string ter↲
minator = ".\n")
```

Parameters:

	`os`	`std::ostream&` for output.

	`terminator`	String to append to the setting words string, default period and newline. Usage:
		`outUncIOFlags(std::cout.iword(1), std::cerr); // uncFlags (0xa08) ↲`
		`add_+/-  adddegfree replicates addlimits`

	`uncFlags`	Output unc class uncertain IO flags value to be displayed as words.

Function outUncIOFlags

boost::quan::outUncIOFlags

Synopsis

```
// In header: <boost/quan/unc.hpp>


void outUncIOFlags(std::ostream & os = std::cout,
                   std::ostream & osout = std::cerr,
                   std::string terminator = ".\n");
```

Description

Show the set uncertain class `std::stream` flags settings as words, for example: "add_/-, set_scaled, addlimits". Usage:

```
outUncIOFlags(std::cout.iword(1), std::cerr); // uncFlags (0xa08) add_+/-  adddegfree replicates ↲
addlimits.
```

See Also:

Version that takes the long uncFlags as parameter.

```
void outUncIOFlags(long uncFlags, std::ostream& osout = std::cerr, std::string terminator = ".\n")
```

Parameters:

	`os`	`std::ostream` whose unc IO flags are to be output.

`osout`	`std::ostream` log to receive output.
`terminator`	`std::string` to be appended to the list of settings, default period and newline.

Function outUncTypes

boost::quan::outUncTypes

Synopsis

```
// In header: <boost/quan/unc.hpp>


void outUncTypes(unsigned short int uncTypes, std::ostream & os = std::cerr);
```

Description

Output word description for each `unc_type` bit.
Usage:

```
outUncTypes(unc.getUncTypes(), std::cerr); // logs to cerr.
```

Parameters:	`os`	`std::ostream&` for output string list of words, default = `std::cerr`.
	`uncTypes`	uncertain types as a short int.

Function outUncValues

boost::quan::outUncValues

Synopsis

```
// In header: <boost/quan/unc.hpp>


void outUncValues(std::ostream & os = std::cout,
                  std::ostream & log = std::cerr);
```

Description

Output ALL the `os.word()` values to the `std::ostream&` log file.

> **Note**
>
> Before call of setuncdefaults() all are zero (probably) and after call: "UncValues: uncFlags 0, setSigDigits 3, uncWidth 10, uncSetWidth -1, uncScale 0, uncSetScale 0, uncUsed 0, uncOldFlags 0, uncOldUncWidth -1, uncOldScale -1, uncSigDigits 2, uncoldSigDigits -1, oldUncUsed -1, oldStdDevSigDigits -1, setUncSigDigits 2, roundingLossIndex 0.05, confidenceIndex 0.05.\n";

Parameters:	`log`	`std::ostream&` whose unc values are to be sent to log.
	`os`	Current `std::ostream&` to be displayed in log.

Function plusminus

boost::quan::plusminus — Set `std::ostream` so that uncertainty estimate, usually standard deviation, is added when for output of **value** (mean) of class `unc`. This is NOT the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & plusminus(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << plusminus << u ...
```

outputs a mean value and standard deviation : "1.23 +/- 0.05"

Parameters:	`iostr` `std::ostream` for which values should add an uncertainty estimate.
Returns:	`std::ostream` to make chainable.

Function scale

boost::quan::scale — Set `std::ostream` so that scaling of output values to a preset factor takes place for output of **value** (mean) of class `unc`. This is NOT the default.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & scale(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
std::cout << scale << u ...
```

< set bit 1 = 1 to mean scaled.

Parameters:	`iostr` `std::ostream` for which values should be scaled by the preset factor.
Returns:	`std::ostream` to make chainable.

Function setsigdigits

boost::quan::setsigdigits — Set `std::ostream` so that the number of significant digits (precision) for output of **value** of class `unc` is controlled by the set precision.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & setsigdigits(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
<< setsigdigits ...
```

Will use precision computed from a previously set precision for this stream.

Parameters:	`iostr`	`std::ostream` for which to set precision to show value.
Returns:	`std::ostream` to make chainable.	

Function setUncDefaults

boost::quan::setUncDefaults — Random id value bracketing the various uncertain class display attributes.

Synopsis

```
// In header: <boost/quan/unc.hpp>


void setUncDefaults(std::ios_base & os);
```

Description

Set 16 default flag bit values for uncertain output flags. Usage:

```
setUncDefaults(cout);
```

Warning

`setUncDefaults()` MUST be called before using a stream.

Default iword(*) values are all zero - these are not usable. Function `setUncDefaults()` sets some defaults Can set individual value, for example, for 3 sig Digits, os.iword(sigDigitsIndex) = 3;

Note

Index values must have been initialised by xalloc calls.

Note

Might be best to make `setUncDefaults` also set `std::ios_base` defaults with `setiosDefaults(ostream&)`;?

Function setuncsigdigits

boost::quan::setuncsigdigits — Set `std::ostream` to set the number of significant digits for output of uncertainty of class `unc`, for example: +/- 1.23.

Synopsis

```
// In header: <boost/quan/unc.hpp>


std::ios_base & setuncsigdigits(std::ios_base & iostr);
```

Description

Example: Using `std::ostream` manipulator:

```
<< useSetUncSigDigits(2) ...
```

will use precision of 2 decimal digits for output of uncertainty (standard deviation).

< Use stdDev sigDigits stored with

Parameters:	`iostr`	`std::ostream` for which to set precision to show uncertainty.
Returns:	`std::ostream` to make chainable.	

Function unc_input

boost::quan::unc_input — Uncertainties ARE correlated. This is the unusual case where the sum must be exact, so the uncertainties are subtracted. Example: two blocks which fit into a box perfectly. So if both have an uncertainties, they must cancel when the uncertainties are added. Also applies to items like concentrations which must add up to 100% or unity.

Synopsis

```
// In header: <boost/quan/unc.hpp>


void unc_input(double & mean, double & stdDev,
               unsigned short int & degreesOfFreedom,
               unsigned short int & uncTypes, std::istream & is);
```

Description

Inputs uncertainty as value, (implicitly exact, std deviation = 0). & optionally an explicit measure of uncertainty [[+]|[-] <standard deviation * 2. >], (1.0 implies 1. +|- 0.5 and sd of 0.5, 1.00 implies 1. +|- 0.05 and sd of 0.05)

& optionally degrees of freedom [(<short int>)] like (99) Used by istream& operator>> (istream&, unc<is_correlated>&) Original simple version: char plus, slash, minus; s >> value >> plus >> slash >> minus >> stdDev; if ((plus != '+') || (slash != '/') || (minus != '-')) { cerr << "Unexpected characters encountered in reading " "value +/- stdDev !" << endl; is.setf(ios_base::failbit);

Used by std::istream& operator>> (std::istream& is, const unc }

Function template unc_of

boost::quan::unc_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> double unc_of(double);
```

Description

Returns:	zero if no uncertainty information is available (for built-in double, float, or long double).

Function template unc_of

boost::quan::unc_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> double unc_of(float);
```

Description

Returns:	zero if no uncertainty information is available (for built-in double, float, or long double).

Function template unc_of

boost::quan::unc_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<typename T> double unc_of(T);
```

Description

Allow uncertainty (standard deviation) part of variables of class unc to be assigned to, and compared with float.

Template Parameters:	`T` Built-in floating-point type, float, double or long double, or uncertain type unc.
Returns:	zero if no uncertainty information is available (for built-in double, float, or long double).
Returns:	zero if no uncertainty information is available (for built-in double, float, or long double).

Function template unc_of

boost::quan::unc_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> double unc_of(uncun u);
```

Description

Returns:	zero if no uncertainty information is available (for built-in double, float, or long double).

Function template uncs_of

boost::quan::uncs_of — Get uncertainties (standard deviation) as a pair of const float values.

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<typename T>
  const std::pair< float, float > uncs_of(std::pair< const T, T > vp);
```

Description

Template Parameters:	`T` Builtin-floating point type or unc.
Returns:	uncertainty parts (if any) as a pair of floats.
Returns:	uncs_of uncertainties (standard deviation) as a pair of **float** values.

Function template uncs_of

boost::quan::uncs_of — Get uncertainties (standard deviation) of a pair of values.

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> std::pair< double, double > uncs_of(std::pair< double, double > vp);
```

Description

Function template uncs_of

boost::quan::uncs_of — Get uncertainties (standard deviation) as a pair of const float values.

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<typename T> std::pair< double, double > uncs_of(std::pair< T, T > vp);
```

Description

Template Parameters:	`T` Built-in floating-point type, float, double, long double or unc or `Meas`.
	Builtin-floating point type or unc.
Returns:	uncertainty parts (if any) as a pair of floats.
Returns:	uncs_of uncertainties (standard deviation) as a pair of **float** values.

Function template uncs_of

boost::quan::uncs_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<typename T1, typename T2>
  std::pair< double, double > uncs_of(std::pair< T1, T2 >);
```

Description

Function template uncs_of

boost::quan::uncs_of — Get uncertainties (standard deviation) as a pair of double values.

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> std::pair< double, double > uncs_of(std::pair< uncun, uncun > vp);
```

Description

Returns: uncs_of uncertainties (standard deviation) as a pair of double values.

Function template value_of

boost::quan::value_of — <

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> double value_of(double v);
```

Description

Returns: value as a double.

Function template value_of

boost::quan::value_of — Two helper functions to provide values and uncertainties as pairs.

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<typename T> double value_of(T);
```

Description

<

Note

Names: value_of (single value) and plural valueS_of (pair).

Allow value part of variables of class unc to be assigned to, and compared with double.

Template Parameters:	`T`	Built-in floating-point type, float, double or long double, or uncertain type unc. value type convertible to double.
Returns:	value as a double.	
Returns:	value as a double.	

Function template value_of

boost::quan::value_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> double value_of(unc< false > v);
```

Description

Returns: unc.value() as a double.

Function template value_of

boost::quan::value_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> double value_of(unc< true > v);
```

Description

Returns: unc.value() as a double.

Function template values_of

boost::quan::values_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<>
  std::pair< double, double > values_of(std::pair< double, double > vp);
```

Description

< < <

Returns:	values of a pair of double values.
Returns:	values of a pair of double values.
Returns:	values of a pair of double values.

Function template values_of

boost::quan::values_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<typename T> std::pair< double, double > values_of(std::pair< T, T >);
```

Description

Template Parameters:	`T`	Built-in floating-point type, float, double, long double or unc or `Meas`.

Function template values_of

boost::quan::values_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<typename T1, typename T2>
  std::pair< double, double > values_of(std::pair< T1, T2 >);
```

Description

Function template values_of

boost::quan::values_of

Synopsis

```
// In header: <boost/quan/unc.hpp>


template<> std::pair< double, double > values_of(std::pair< uncun, uncun > vp);
```

Description

< < <

Returns:	values of a pair of double values.
Returns:	values of a pair of double values.
Returns:	values of a pair of double values.

Header <boost/quan/unc_init.hpp>

Initialisation of `std::stream` iword to store uncertainty information.

This sets the index values for data stored using the `std::ios_base::xalloc` mechanism. This allows a memory of stream state from previous calls of a `std::iostream`. This file `unc_init.hpp` is included from `unc.hpp` for many functions. Function `setUn-cdefaults()` is used to set default values for these xalloc items for a specified `std::iostream` (default `std::cout`).

Global confidenceIndex

boost::quan::confidenceIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long confidenceIndex;
```

Global isIndexed

boost::quan::isIndexed — Above indexes have been initialised.

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

bool isIndexed;
```

Global oldScaleIndex

boost::quan::oldScaleIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldScaleIndex;
```

Global oldSigDigitsIndex

boost::quan::oldSigDigitsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldSigDigitsIndex;
```

Global oldUncFlagsIndex

boost::quan::oldUncFlagsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldUncFlagsIndex;
```

Global oldUncSetWidthIndex

boost::quan::oldUncSetWidthIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldUncSetWidthIndex;
```

Global oldUncSigDigitsIndex

boost::quan::oldUncSigDigitsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldUncSigDigitsIndex;
```

Global oldUncUsedIndex

boost::quan::oldUncUsedIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldUncUsedIndex;
```

Global oldUncWidthIndex

boost::quan::oldUncWidthIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldUncWidthIndex;
```

Global oldWidthIndex

boost::quan::oldWidthIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long oldWidthIndex;
```

Global roundingLossIndex

boost::quan::roundingLossIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long roundingLossIndex;
```

Global scaleIndex

boost::quan::scaleIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long scaleIndex;
```

Global setScaleIndex

boost::quan::setScaleIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long setScaleIndex;
```

Global setSigDigitsIndex

boost::quan::setSigDigitsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long setSigDigitsIndex;
```

Global setUncSigDigitsIndex

boost::quan::setUncSigDigitsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long setUncSigDigitsIndex;
```

Global sigDigitsIndex

boost::quan::sigDigitsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long sigDigitsIndex;
```

Global topIndex

boost::quan::topIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long topIndex;
```

Global uncFlagsIndex

boost::quan::uncFlagsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long uncFlagsIndex;
```

Global uncSigDigitsIndex

boost::quan::uncSigDigitsIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long uncSigDigitsIndex;
```

Global uncWidthIndex

boost::quan::uncWidthIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long uncWidthIndex;
```

Global usedIndex

boost::quan::usedIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long usedIndex;
```

Global widthIndex

boost::quan::widthIndex

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long widthIndex;
```

Global zeroIndex

boost::quan::zeroIndex — This block of definitions MUST be positioned before function main() is called. 14 indexes of long iwords allocated by calls of std::ios_base::xalloc(); 1st call of xalloc() returns 0 so std::ios_base.iword(0) used for magic ID, 2nd call of xalloc() returns 1 so std::ios_base.iword(1) used for uncFlags, 3rd calls returns 2, so iosword(2) used for sigDigits ... Order of assignment must ensure these match enum uncindex (if used). zero_index and top_index are used to hold a 'magic' unique number to show that the `std::ios_base::xalloc` values have been initialised, and not corrupted.

Synopsis

```
// In header: <boost/quan/unc_init.hpp>

const long zeroIndex;
```

Header <boost/quan/xiostream.hpp>

Extra iostream manipulators and output of stream state descriptions.

Paul A. Bristow

Sep 2009, 2021

```
namespace boost {
  namespace quan {
    class chars;
    template<typename T> class oapp;
    template<typename T> class omanip;
    class setupperbase;
    class stars;

    const char * fmtFlagWords;
    template<typename T> std::ostream & FPclass(std::ostream & os, T value);
    std::ios_base & hexbase(std::ios_base &);
    std::ios_base & lowercase(std::ios_base & _I);
    std::ostream & operator<<(std::ostream &, const chars &);
    template<typename T>
      std::ostream & operator<<(std::ostream & os, const omanip< T > & m);
    std::ostream & operator<<(std::ostream &, const stars &);
    void outFmtFlags(std::ios_base::fmtflags = std::cout.flags(),
                     std::ostream & = std::cerr, const char * = ". ");
    template<typename T> void outFpClass(T, std::ostream &);
    void outIosFmtFlags(long flags, std::ostream & os);
    void outIOstates(std::ios_base::iostate = std::cout.rdstate(),
                     std::ostream & = std::cerr, const char * = ". ");
    void setiosDefaults(std::ostream & os);
    std::ostream & showformat(std::ostream &);
    std::ostream & showiostate(std::ostream &);
  }
}
```

Class chars

boost::quan::chars — Manipulator class to help output int repeat count & character chars. Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


class chars {
public:
  // construct/copy/destruct
  chars(int, char);

  // friend functions
  std::ostream & operator<<(std::ostream &, const chars &);
};
```

Description

```
out << chars(10, '_') .... for 10 underlines.
```

`chars` public construct/copy/destruct

1.
```
chars(int n, char c);
```

`chars` friend functions

1.
```
std::ostream & operator<<(std::ostream &, const chars &);
```

Manipulator to help output int repeat count & character chars. Example:

```
out << chars(10, '_') .... for 10 underlines.
```

Class template oapp

boost::quan::oapp

Synopsis

```
// In header: <boost/quan/xiostream.hpp>

template<typename T>
class oapp {
public:
  // construct/copy/destruct
  oapp(std::ostream &(*)(std::ostream &, T));

  // public member functions
  omanip< T > operator()(T);
};
```

Description

oapp public construct/copy/destruct

1.
```
oapp(std::ostream &(*)(std::ostream &, T) f);
```

oapp public member functions

1.
```
omanip< T > operator()(T v);
```

Class template omanip

boost::quan::omanip

Synopsis

```
// In header: <boost/quan/xiostream.hpp>

template<typename T>
class omanip {
public:
  // construct/copy/destruct
  omanip(std::ostream &(*)(std::ostream &, T), T);
};
```

Description

`omanip` public construct/copy/destruct

1.
```
omanip(std::ostream &(*)(std::ostream &, T) f, T v);
```

Class setupperbase

boost::quan::setupperbase — class setupperbase to output uppercase or capital letter B, X and O when using base other than 10

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


class setupperbase {
public:
  // construct/copy/destruct
  setupperbase(int);
};
```

Description

`setupperbase` public construct/copy/destruct

1.
```
setupperbase(int base);
```

Class stars

boost::quan::stars — Manipulator class to help to output a number of stars. Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


class stars {
public:
  // construct/copy/destruct
  stars(int);

  // friend functions
  std::ostream & operator<<(std::ostream &, const stars &);
};
```

Description

```
out << stars(10) ...
```

`stars` public construct/copy/destruct

1.
```
stars(int n);
```

`stars` friend functions

1.
```
std::ostream & operator<<(std::ostream &, const stars &);
```

   Manipulator to output a number of stars. Example:

```
out << stars(10) ...
```

Global fmtFlagWords

boost::quan::fmtFlagWords

Synopsis

```
// In header: <boost/quan/xiostream.hpp>

const char * fmtFlagWords;
```

Function hexbase

boost::quan::hexbase — Function to set base to std::hex & std::showbase & std::uppercase too. Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


std::ios_base & hexbase(std::ios_base & _I);
```

Description

```
out << hexbase << ... for 1234ABCD
```

equivalent to

```
std::cout << std::hex << std::showbase << std::uppercase ...
```

Function operator<<

boost::quan::operator<< — Manipulator to help output int repeat count & character chars. Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


std::ostream & operator<<(std::ostream & os, const chars & s);
```

Description

```
out << chars(10, '_') .... for 10 underlines.
```

Function operator<<

boost::quan::operator<< — Manipulator to output a number of stars. Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


std::ostream & operator<<(std::ostream & os, const stars & s);
```

Description

```
out << stars(10) ...
```

Function outFmtFlags

boost::quan::outFmtFlags

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


void outFmtFlags(std::ios_base::fmtflags fmtFlags = std::cout.flags(),
                 std::ostream & os = std::cerr, const char * term = ". ");
```

Description

Output string descriptions of `std::ios::fmtflags`.

For example, logs to std::cerr"FormatFlags: skipws showbase right dec"

```
void outFmtFlags(fmtflags fmtFlags = cout.flags(), std::os↵
tream& os = cerr, const char* term = ".\n");
```

Function template outFpClass

boost::quan::outFpClass — Custom outputs for non-finite values NaN, inf ... (rather than Microsoft default "1#IND ...") Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


template<typename T> void outFpClass(T, std::ostream &);
```

Description

```
outFpClass(x, std::cerr);
```

Template Parameters:	`T`	Floating-point not-finite type (usually `double` or `float`) to be described.

Function outIOstates

boost::quan::outIOstates — Output all std::ios iostates.

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


void outIOstates(std::ios_base::iostate rdstate = std::cout.rdstate(),
                 std::ostream & os = std::cerr, const char * term = ". ");
```

Description

Usages: Default logs `std::cout` iostate to `std::cerr`, for example "IOstate: good", or "IOstate: fail"

```
outIOstates(); // Same as:
outIOstates(cout.rdState(), cerr, ".\n");
outIOstates(cin.rdState());
outIOstates(cerr.rdState(), cout, ", ");
outIOstates(cout.rdState(), cerr, " iostate.\n ");
```

Function showformat

boost::quan::showformat — Show IO stream format flags descriptions in words for this stream. Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


std::ostream & showformat(std::ostream & os);
```

Description

```
std::cout << showformat  ...
```

Function showiostate

boost::quan::showiostate — Show std::IO stream state in descriptive words for this stream. Example:

Synopsis

```
// In header: <boost/quan/xiostream.hpp>


std::ostream & showiostate(std::ostream & os);
```

Description

```
std::cout << showiostate ...
```

Indexes

Class Index

Function Index

typedef Index

Symbols

Index