Robust Services Core

Coding Guidelines

Version 1.3

July 24, 2017

CONTENTS

FORMATTING

1. Begin each file with the following heading:
```
//=============================================================================
//
//  <FileName>
//
//  Copyright (C) 201n-201n <Name>.  All rights reserved.
//
```
2. Use spaces instead of tabs.
3. Indent a multiple of 3 spaces.
4. Remove unnecessary interior spaces and semicolons. Remove trailing spaces.
5. Use `//` comments instead of `/*...*/`.
6. Add blank lines for readability, but avoid multiple blank lines.
7. Limit lines to 80 characters in length. When breaking at punctuation, break after `)`, and before `:(`. When breaking at an operator, the break can occur before or after, depending on what reads better.
8. Almost always use Camel case. Use uppercase and underscores only in low-level types and constants. Names that evoke Hungarian notation are an abomination.
9. Keep `*` and `&` with the type instead of the variable (`Type* t` instead of `Type *t`).

INTERFACES

1. Insert an `#include` guard based on the file name (`filename.ext` and `FILENAME_EXT_INCLUDED`) immediately after the standard heading.
2. Sort `#include` statements as follows:
   a. the header that defines the base class of the class defined in the header
   b. C++/C library headers, in alphabetical order
   c. other headers, in alphabetical order
3. Remove an `#include` solely associated with functions inherited from a base class.
4. Remove an `#include` by forward declaring a class that is only named in references or pointers. Use an explicit forward declaration instead of relying on this as a side effect of a friend declaration.
5. Remove `using` declarations and directives. Prefix the namespace directly (i.e. `std::`<symbol>).
6. Initialize global data (static members) in the .cpp if possible.

IMPLEMENTATIONS

1. Order `#include` statements as follows:
   a. the header that defines the functions being implemented
   b. C++/C library headers, in alphabetical order
   c. other headers, in alphabetical order
2. Omit any `#include` or `using` that is already in the header.
3. Put all of the code in the same namespace as the class defined in the header.
4. Implement functions alphabetically, after the constructor(s) and destructor.
5. Separate functions in the same class with a `//----...` that is 80 characters long.
6. Separate private classes (local to the .cpp) with a `//====...` that is 80 characters long.
7. Add a blank line after the `fn_name` that defines a function's name for `Debug::ft`.
8. Name constructors and destructors "<class>`.ctor`" and "<class>`.dtor`" for `Debug::ft`.
9. Fix all compiler warnings.

CLASSES

1. Give a class its own .h and .cpp unless it is trivial, closely related to others, or private to an implementation.
2. A base class should be abstract. Its constructor should therefore be *protected*.
3. Tag a constructor as explicit if it can be invoked with one argument.
4. Make each public function non-virtual, with a one-line invocation of a virtual function if necessary.
5. Make each virtual function private if possible, or protected if derived classes may need to invoke it.
6. Make a base class destructor
    a. virtual and public
    b. non-virtual and protected
    c. virtual and protected, to restrict deletion
7. If a destructor frees a resource, even automatically through a `unique_ptr` member, also define
    a. a copy constructor: `Class(const Class& that);`
    b. a copy assignment operator: `Class& operator=(const Class& that);`

    If the class allows copying, also define
    c. a move constructor: `Class(Class&& that);`
    d. a move assignment operator: `Class& operator=(Class&& that);`
    - In C++11, each of the above function can be suffixed with *delete* to prohibit its use, or *default* to use the compiler-generated default. The pre-C++11 equivalents are to make the function private (*delete*) or not declare it at all (*default*).
    - In a copy assignment operator, create copies of `that`'s resources first, then release `this`'s existing resources, and finally assign the new ones.
    - A "move" function is an optimization that releases the existing resources and then takes over the ones owned by `that`. Its implementation typically uses `std::swap`.
8. To prohibit stack allocation, make constructors private, and/or make the destructor private.
9. To prohibit scalar heap allocation, define *operator new* as private.
10. To prohibit vector heap allocation, define *operator new[]* as private.
11. If a class only has *static* members, convert it to a namespace. If this is not possible, prohibit its creation.
12. Include *virtual* and *override* when overriding a function defined in a base class.
13. Make a function or argument *const* when appropriate.
14. Remove *inline* as a keyword.
15. Avoid *friend* where possible.
16. Override `Display` if a class has data.
17. Override `Patch` except in a trivial leaf class.
18. Avoid invoking virtual functions in the same class hierarchy within constructors and destructors. Provide an implementation for a pure virtual function to highlight the bug of calling it too early during construction or too late during destruction.
19. If a class is large, consider using the PIMPL idiom to move its private members to the .cpp.
20. When only a subset of a class's data should be write-protected, split it into a pair of collaborating classes that use `MemProt` and `MemDyn` (or `MemFirm` and `MemPerm`).
21. Static member data begins with an uppercase letter and ends with an underscore, which may be omitted if it is not returned by a "`Get`" function. Non-static member data begins with a lowercase letter and ends with an underscore, which may be omitted if the field is often used externally, as in a struct.

FUNCTIONS

1. Use the initialization list for constructors. Initialize members in the order that the class declared them.
2. Use `()` instead of `(void)` for an empty argument list.
3. Name each argument. Use the same name in the interface and the implementation.
4. Make the invocation of `Debug::ft` the first line in a function body, and follow it with a blank line.
5. The `fn_name` passed to `Debug::ft` and similar functions should accurately reflect the name of the invoking function.
6. Trace a "`Get`" function only if it is virtual.
7. Left-align the types *and* variable names in a long declaration list.
8. Use `nullptr` instead of `NULL`.
9. Check for `nullptr`, even when an argument is passed by reference. (A reference merely *documents* that `nullptr` is an invalid input.)
10. After invoking `delete`, set a pointer to `nullptr`.
11. Use `unique_ptr` to avoid the need for `delete`.
12. Use `unique_ptr` so that a resource owned by a stack variable will be freed if an exception occurs.
13. Declare loop variables inline (e.g. `for auto i =`).
14. Declare variables of limited scope inline, as close to where they are used as is reasonable.
15. Use `auto` unless specifying the type definitely improves readability.
16. Include `{...}` in all non-trivial `if`, `for`, and `while` statements.
17. Use braces in both or neither clause of an `if-endif`, even if one clause is a single statement.
18. When there is no `else`, use braces for the statement after `if` unless it fits on the same line.
19. Define string constants in a common location to support future localization.
20. To force indentation, even in the face of automated formatting, use `{...}` between function pairs such as
    a. `EnterBlockingOperation` and `ExitBlockingOperation`
    b. `Lock` and `Unlock`
    c. `MakePreemptable` and `MakeUnpreemptable`

TOOLS

PARSE

- output location where parse failed, along with source code and pointer (caret)
  - `Punt()` could provide a diagnostic; what about `Retreat()`?
- support `#define`
  - `define.ini` file for initial `#defines`
  - `#define` *symbol* (equates to blanks, but can be used by `#ifdef`)
  - `#define` *symbol n* (numeric constant, equivalent to `const auto symbol = n`)
  - `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif` for conditional compilation
- support anonymous namespaces
- support argument-dependent lookup in a class's scope
  - `getline` currently requires `std::` prefix to be resolved but should find the version in `<string>`
  - `next` shouldn't need `std::` prefix, because iterator is already in `std`
- support argument-dependent lookup in an enum's scope
  - `strAccess` and `strClassTag` shouldn't need `Cxx::` prefix to compile
- support "`= delete`" and "`= default`" for constructor, destructor, and assignment operator
  - convert private functions to "`= delete`" once supported (may want to wait, as it is pure C++11)
- support scoped enums
  - modify some enums at namespace scope to use them
  - converting from a scoped enum to an int requires a static_cast, so don't convert an enum that primarily serves as a const int
- support ..." <blanks> "... as the continuation of a string literal
- operator overloads are not captured as usages, because `Operation`'s operator has no referent
- `Numeric.ctor` is logged as unused but is actually used by brace initialization
- `auto c = Class()` would not find a `Class.operator=` overload
- inline friend function is not executed for a template instance
  - the function does not belong to the template, so it is not executed during instantiation
  - the solution might be similar to that for `Singleton.Instance_`, which should also be executed for each specialization but is bypassed because it is not defined inside the template class
- Singleton instances log `Instance_` as uninitialized
  - occurs because these statements are not executed
- have `StackArg` track lvalues and rvalues so that `Function.CanBeInvokedWith` can consider them when selecting a function
- `MatchWith(char, int)` invoked for `INT8_MIN = -128`; result is `Abridgeable`
  - track that a literal is being assigned so that `Compatible` can be returned, averting a search for conversion operators
  - `IntLiteral` uses `Terminal` (int) as its `ResultType`, but should probably use itself so that its value will be available
  - `-128` is parsed as a unary minus with a 128 literal: either parse `-128`as a literal or apply the unary minus to the literal in `Operator.Execute` in order to update it
  - `IntLiteral.Referent` could look at literal and return char or short if it's within the min-max range
- `Punt()` causes a software log on argument overflow
- fix "assume template instance is visible" kludge in `CxxScoped.NameRefersToItem`
- add template class instances to the correct scope instead of burying them in template classes
  - note that template function instances are added to the correct scope
- check that memory usage is the same after *parse*, *restart warm*, *parse*
- split up `AssignedTo` by defining `WasAssigned`, `WasPassed`, and `WasReturned`
- integrate `BlockCopied` with `WasWritten`
  - copying of a second-level nested class is probably not handled correctly

  - invoke `WasPassed` instead of `WasWritten` when an item is passed to a non-const pointer or reference argument; also invoke something on the argument, in case it block-copied a class (however, the argument doesn't want to increment writes_, since calls_ already tracks writes indirectly)
- o track file and line number for each init/read/write/call
- o add option to include source code line numbers in execution trace
- o `Scope` and `clear` are repeated in execution trace
- x invoke `IncrCalls` on constructors up the class hierarchy
- x support multiple forward declarations of functions as well as classes: currently, only one of the function declarations would probably be considered implemented [no: a non-member function is usually defined in a low-level header that can be `#included` without dragging in too many symbols]
- x make local enums and typedefs true locals instead of temporary globals [no: these are rare in functions, so doing this would add little value]

CHECK

- o `StackArg.via_` incorrectly flagged as "data could be const"
  - caused by `auto token = (index == 0 ? item : via_)`
  - `item` sets type for `token` (non-const `CxxToken*`), is pushed as result, and is assigned to `token`
  - `via_` would be flagged as non-const if copied to the non-const `token`, but this does not occur
- o `parm` incorrectly flagged as "argument could be const" in `void DeleteParm(TlvParmLayout& parm)`
  - occurs because `parm.header.pid` is LHS of an assignment, but a single `via_` can't track a chain of "."
- o `arg` incorrectly flagged as "argument could be const" in `main_t EnterThread(void* arg)`
  - no longer occurs, but cannot be const because `EnterThread` (function signature) is passed as an argument
- o create a configuration file for
  - suppressing specific warnings
  - defining file header (e.g. copyright notice)
  - classes that are exempt from implementing `Display`
  - classes that are exempt from implementing `Patch`
  - functions that are exempt from invoking `Debug::ft`
- o add logs for
  - pointer-int-bool conversions: PointerAsBool, IntAsBool, FalseAsNullptr, ZeroAsNullptr, NullptrAsZero
  - assigning the result of `new` to a stack or member pointer instead of a `unique_ptr` or `shared_ptr`
  - redefining an inherited default argument value [Myers 1.38]
  - returning a handle to internal data [Myers 1.29]
  - preprocessor directive other than `#include` guard
    - parse preprocessor directives
    - parse an `#include` guard: `#ifndef` <guard> `#define` <guard> `#endif`
    - change `NO_OP` to `#define NO_OP` once `#define` is supported
  - copy constructor does not invoke base class copy constructor
    - default is to invoke base class constructor, not copy constructor
  - lines that could be merged and still fall within 80-character limit
  - using statement not in standard order (sort alphabetically)
  - overridden function not in standard order (sort alphabetically)
  - member omitted by `Display`
  - class documentation that does not appear immediately above the class declaration
    - search on "\{[\n]//"
    - there should be no comment after the "{" that begins the class declaration
  - double rule (`//==`=) not used between functions in different classes
  - single rule (`//---`) not used between functions in the same class

TRIM

- various .cpps: +memory, set, string, vector, bitset...
  - occurs even when .h defines a member using a typedef for the STL class
  - allow definition of a chain (e.g. sstream-string-istream-ostream-ios...), in which subsequent items do not have to be included?
- recommendations for a .cpp should take the .h recommendations into account
  - a common example is adding (removing) an `#include` from a .h, which the .cpp must then remove (add)
  - when adding an `#include` to a .h, remove it from .cpp(s)
  - when removing an `#include` from a .h, ask if it should be added to .cpp(s)
- *apply* should provide a way to preserve an `#include`, `using`, or forward declaration that *trim* would modify incorrectly
- `CodeFile.h`: `-CxxNamed.h`, `-CxxScoped.h`
  - needed to make `CodeFile` members `IncludePtr` and `UsingPtr` visible to `unique_ptr`
  - `IncludePtr` is not instantiated until a .cpp #includes `CodeFile.h`, at which time `incls_.push_back` probably has a clause (e.g. failure to expand vector) in which it deletes an `IncludePtr`
  - definition of `Include` must then be visible so that the `unique_ptr` can delete it
  - same occurs for `UsingPtr` and `Using`
  - a template may need to use a private function (e.g. `unique_ptr` using a destructor), but this cannot be detected if the template is in an extern file that provides function declarations, but not definitions
- `TlvMessage.h`: `-Debug.h`
  - occurs because template functions are not executed
  - possible solutions are
    - execute templates to track usage, which will require enhancements to avoid logging errors caused by template parameters not having referents
    - when template is instantiated, add usages to the file that provided its source code, except for those associated with template arguments
- `StreamRequest.cpp`: `-sstream`
  - needed to make `ostringstream` a complete type for deletion
  - occurs because `ostringstream` is deleted by a `unique_ptr`, whose destructor invocation is not executed
- `CodeTypes.h`: `-SysTypes.h`
  - `using namespace NodeBase` then caused a compile error because `NodeBase` was no longer defined
  - after determining which files to `#include`, check using directives to verify that some file being transitively included actually defines an item in that namespace
- `CxxString.h` had a compile error because `using namespace NodeBase` was absent; had been picking it up from `CodeTypes.h`, which had been told to delete it
  - for each symbol resolved by a using statement, check that the file contains a using statement that resolves that symbol

FORMAT

- remove trailing blanks
- remove extra blank lines
- replace tabs with spaces
- indent three spaces
- insert separators and blank lines
- 80-column formatter for code inside functions and brace initialization lists
  - column width can also be exceeded elsewhere
  - primary problem is incorporating comments

OTHER ENHANCEMENTS

- add options to *export* to
  - suppress statistics (reads, writes, calls...) that create noise in diffs
  - generate namespace view
  - generate file view
  - generate class hierarchy
- tool to assist with decoupling
  - what symbols does A use from B?
  - if enums and typedefs in A were relocated, what would be decoupled?
  - look at Lakos's decoupling strategies
- make namespace view compilable by emitting items in their original order, or at least in a way that prevents forward references that cause “undefined symbol” errors
- command to create file containing all functions traced by `Debug::ft`
- command to generate global cross-reference
- library operators `&= -= |=` to support `>assign a -= b` instead of `>assign a (a - b)`
- dependency graph
  - for generating a PDF within C++, see the open source library *libHaru* (aka *Haru*)
- reduce memory usage (idle=52MB, max=320MB, curr=230MB)
- integrate *parse*, *check*, and *trim*
  - one issue is that a "warning" to add an `#include` or forward doesn't have a line number but does have an associated string
  - options might be
    - filename (mandatory)
    - P: generate parse trace
    - p: generate parse trace on failure (filename.parse.txt)
    - x: generate execution trace (filename.obj.txt)
    - w: generate warning file (filename.check.txt)
    - l: generate library file (= "nfh" options) (filename.lib.txt)
    - n: generate namespace view
    - f: generate file view
    - h: generate class hierarchy

TRIM: NOTES AND COMPLETED CHANGES

```
v implement command for applying recommendations
v break up extern files to mimic the structure of external headers, which would
  allow them to be trimmed
v ThreadRegistry.h: +SysMutex.h, -Thread forward declaration: SysMutex is
  transitively #included; is it dubious to #include it only for its forward
  declaration of Thread* [no; could be decoupled later]
v various: +TraceRecord.h
  --suggested because of TraceRecord::Id, which subclasses use as "Id": this
    is inherited, directly or indirectly, so an #include is unnecessary
v IpBuffer.cpp: -SysTcpSocket.h
  --caused a compiler error because result of GetSocket (SysTcpSocket*) was
    assigned to a SysSocket*, which means that SysTcpSocket must be #included
    to ascertain that it is a subclass of SysSocket [since fixed by invoking
    CompatibilityWith for lhs and rhs of assignments]
x when suggesting the addition of a forward declaration, look for an existing
  (courtesy) forward declaration if the symbol is not in the same namespace as
  its user [no: the courtesy declaration will be found if it is transitively
  #included; if it isn't, then it isn't clear that it should be #included]
x in >trim report, include friend declarations that have no users; if friend
```

```
  is external, such as deleter_type, indicate that it may actually be needed
  [no: already included in >check report]
x when showing which items need qualification to remove using statements, break
  up full template instance names into their components [no: cosmetic]
x when showing which items need qualification to remove using statements,
  display aliases (e.g. suggests qualifying SwitchStub::PortId when code
  reads Switch::PortId)
  --generating Switch::PortId is difficult because no item has this as its
    scoped name
  --QualName.class_ = Switch, so perhaps it could override of ScopedName to
    generate the alias
  --this would involve merging the alias with the true scoped name
  --[no: showing the true name may actually be useful]
x flag using statement that .cpp repeats from .h (a base usage file)
  --dubious because it discourages .h from removing using statements? (counter
    argument is that it is no more dubious that logging an #include that .cpp
    repeats from .h, because this discourages .h from trimming its #includes)
  --guideline: OK to rely on a using statement in your own .h but not another
  --[no: difficult to implement because it requires knowing base class files,
    which only >trim calculates]
x some base class constructors are not shown as direct usages; only the base
  class is a direct usage [no: this is arguably correct, shouldn't have a
  negative effect, and trying to "fix" it causes problems because auto v = C()
  fails to find a referent for C when looking for a function, as it expects to
  find C::C, not C]
x a derived class can use a base class item without namespace qualification,
  the only exception being when it initially names its base class
x do not show members as direct usages: only the class itself needs to be
  shown [no: members still shown because this is useful as a reference]
v to document reliance on a forward declaration, >trim suggests #including
  the file that defines it, even if that file is transitively included,
  unless the file contains a base class (even a transitive base class)
v do not need to #include item defined in
  --indirect base class
  --base class of #included class [dubious?]
v investigate if name of friend in another namespace can be qualified (yes,
  but must be forward declared first: can only inject a name into the current
  namespace)
x have >check log unused forward and using declarations [no: those that don't
  resolve symbols should nonetheless be kept if the file might require them;
  this is better than relying on #include dependencies that can change, and
  >trim does a better job of analyzing #includes, usings, and forwards]
v do not suggest adding a forward declaration if
  v the item is defined in a file that will be #included, even transitively
  v the forward declaration already exists in
  v this file: this can occur when the forward declaration resolves a function
    *declaration's* argument that is subsequently replaced by the argument
    parsed in the function's *definition*: in the definition file, the item
    may be directly visible, and so the forward declaration is not inserted
    in symbols[ForwardUsage], and its existence is unknown
  v a file that will be #included
  x a file that is used transitively, and can therefore be recommended to be
    #included [this is dubious because there may not be a true dependency on
    that file, in which case it should not be included simply to access its
    forward declaration]
v ignore a template instance for usage purposes if added to the same scope as
  its template, because its template argument(s) only need to be visible at
  the point where the instance is used
```

```
v decide how to handle forward declarations:
  --relying on Singleton.h as a transitive #include is dubious because there
    is no hard dependency (other than to replace the forward declaration)
  --including NbTypes.h, which is *already* included transitively, allows the
    forward declaration to be removed, and NbTypes.h may even be providing its
    forward as a courtesy
                       Singleton forward  include NbTypes.h  Affecters
  CfgParmRegistry      add                remove             NbTypes, Singleton
  CinThread            add                remove             NbTypes, Singleton
  NbModule             remove             add                NbTypes
  ObjectPoolRegistry   remove             included           NbTypes
  PosixSignalRegistry  remove             add                NbTypes
  SysSignals           remove             add                NbTypes
v when using a typedef of a template instance, do not suggest replacing it with
  a forward declaration
v need to forward declare a template class if it isn't visible (e.g. Singleton)
v execution treats typedef transparently, but >trim cannot (hence DirectType)
v do not suggest qualification for an item in the same namespace
v do not suggest adding a forward declaration for an external item
v suggest qualification for a template instance whose template class is in
  another namespace
v NbTracer.h
  --suggests removing the forward declaration of IpBuffer, which *is* used
    indirectly but is also declared in NbTypes.h, which is #included (and
    which is the forward declaration that resolves IpBuffer)
v TcpIoThread.h
  --suggests including SysIpL3Addr.h to pick up forward declaration of
    SysTcpSocket: this is OK, because that is the forward declaration that
    resolved SysTcpSocket, so TcpIoThread.h must already use it transitively
x analyze overridden functions, flagging their usages as "inherited": including
  or declaring forward anything that is inherited is unnecessary (but it isn't
  wrong, and removing it won't eliminate any dependencies, so leave it alone)
x types that underlie public and protected data members are also inherited, but
  derived classes need to allow transparent NUC changes in base classes;
  consequently, only types in virtual functions should be considered inherited
x the way #includes and forwards are analyzed is file-based but should probably
  be class-based, because a file can define multiple classes [break the file up
  if you want this!]
x log reliance on a forward declaration in another header [no: one practice
  is to define a header with courtesy forward declarations, such as <iosfwd>]
```

CLOSED/NOTES

```
v invocations of GetExpression:
  Function          Delimited by  Force    Purpose
  GetArgument       ,;              Y      default argument value
  GetArgList        ,)              Y      argument value
  GetArraySpec      ]               Y      array size
  GetBraceInit      ,}              Y      field value
  GetCase           :               Y      case label
  GetCast           outer expr      -      expression to be cast
  GetClassData      ;=              Y      field width
  GetClassData      ;               Y      initialization value
  GetConditional    :               Y      value if true
  GetConditional    outer expr      -      value if false
  GetDelete         outer expr      -      pointer to delete
  GetEnumMember     ,}              Y      member value
```

```
  GetExpr           ;               -      assignment, function call, null statement
  GetFileData       ;               Y      initialization value
  GetFor            ;               Y      initial clause
  GetFor            ;               Y      conditional clause
  GetFuncData       ;               Y      initialization value
  GetParExpression  )               Y      parentheses
  GetReturn         ;               Y      return value
  GetSubscript      ]               -      index value
  GetThrow          outer expr      -      throw value
v when a unique_ptr is instantiated in a .cpp (but not in a .h), the nullptr
  default parameter for reset() is left on the stack, causing an #ERR! in clear
  --in a .h, no other code is being executed, so operator stack is empty when
    Context.Execute is invoked, which discards the argument on top of the stack
  --in a .cpp, other code is being executed, so operator stack is *not* empty
    when Context.Execute is invoked, which leaves that argument on the stack
  --fixed by unwinding argument stack when popping parser
v verify that operands are compatible (lhs and rhs in assignment, for example)
x template argument need not be inaccessible in scope where used [documented]
x templates can be instantiated because of code in headers: should only occur
  when compiling .cpps, which might make it easier to resolve some >trim bugs
  [no: a header should compile on its own, as if a .cpp using only the header
  #included it, which is what instantiating the template simulates]
v no usage for varSet_ found, but Library.Startup writes to it ["no usage"
  means no readers; this does not necessarily mean (as is the case here)
  that the item does nothing useful]
x shrink_to_fit does nothing: can only account for ~70MB out of ~180MB, even
  when using capacity() instead of size() to get string and vector sizes
x auto v2 = vector<T>(v1.begin(), v1.end()).swap(v1) is the idiomatic way
  to replace a vector with one of the desired size [not implemented because
  containers do not appear to have a lot of unused space]
x optimize an Expression that contains one item (e.g. a Literal or QualName)
  [no: memory savings would be modest]
x convert strings to char* (need option to recreate string) [no: memory
  savings be modest]
x "auto*" is displayed (even though the source code reads "auto") when the type
  is a pointer [it isn't actually wrong and occurs because the ptrs_ field can
  be modified, so it no longer contains the value that was parsed]
x if a template class returns T&, does it make sense to return arg*& when the
  argument is tagged with a pointer, or should the "&" be deleted? [no: keep it]
x pair<string*,bool> and pair<CxxNamed**,bool> are not instantiated (these are
  the return type from hash table insert functions, but are not used)
x capture #define as an identifier (e.g. Posix signal), but not the #define of
  #include guards, as they have no value (a value of EMPTY_STRING, literally)
  [changed Posix signals from #define to const signal_t SIGname = n]
x EnterBlock is invoked twice on Data components (e.g. array spec) when
  UpdateDeclaration returns true [legitimate, because both .h and .cpp use it]
x subclass CxxToken (and other library items) from nothing [had a negligible effect
  on speed or memory usage]
```