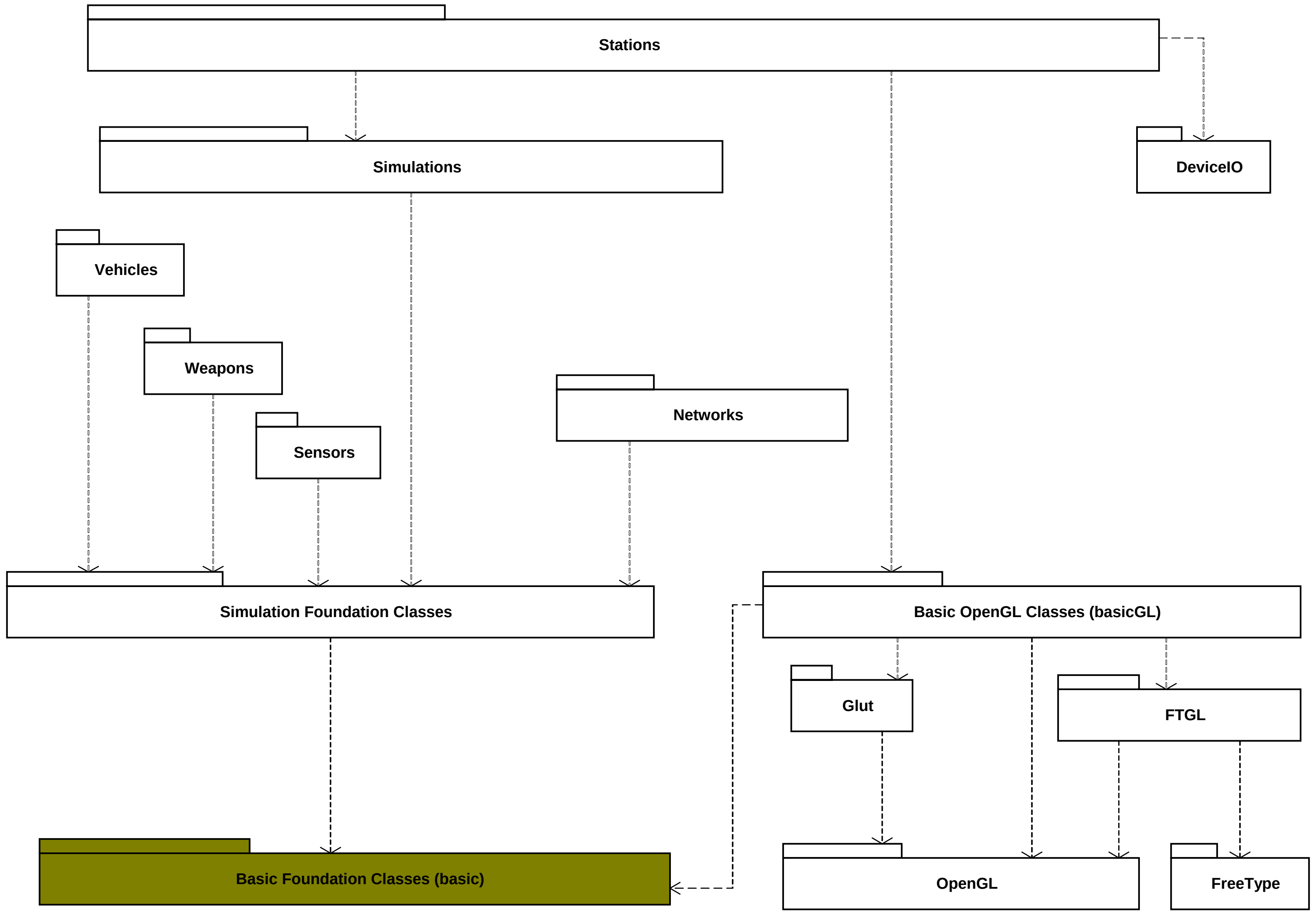
OpenEaagles
Simulation Framework

Basic Foundation Classes

Revised: Nov 14, 2006

Software Layers & Packages


Stations
Simulations
DeviceIO
Vehicles
Weapons
Sensors
Networks
Simulation Foundation Classes
Basic OpenGL Classes (basicGL)
Glut
FTGL
Basic Foundation Classes (basic)
OpenGL
FreeType

Some Design Philosophies

- Open Standards & Packages: C++, OpenGL, Glut, FreeType, FTGL
  - Leaving room to upgrade from C++ to Java or C#

- Object Oriented
  - Common Component Interfaces (Plug-Ins)
    - Consistency within OpenEaagles native components
    - Wrappers for non-native components
  - Lists and sub-lists of components (objects)
  - Push (vs. Pull) method of data flow
    - Example: Sending (pushing) data to a lists of subcomponents vs. subcomponents locating (pulling) data from their container
    - Example: an “update data” event sent to a graphic subcomponent
  - Lots of Polymorphism

Performance

- Real-Time performance for virtual stations

- General Real-Time Rules
  - Tasks must be completed within given time constraints
    - Minimum frame rates (e.g., Nyquist Theorem, which states that the highest frequency which can be accurately represented is less than one-half of the sampling rate)
    - Maximum system latency
  - Threads (processes) can not wait
  - Perform time critical tasks and return
  - Use unblocked (no wait) I/O
  - Use member variables to keep track of the state of a task from one frame to the next (e.g., state machines)

- Two primary threads (Dual processor PC model):
  - Real-time thread (time critical calculations)
  - Background thread (graphics & non-time critical tasks)

- Tradeoff between Flexibility and Performance
  - Note: The exponential increase in CPU performance on low cost PC based systems is NOT a myth.

Open System

- Operating Systems: Windows, Linux (and MAC OS-X)
- Processors: Intel Pentium, AMD (and PowerPC)
- Compilers: Microsoft Visual Studio, GCC
- Architectures: 32 Bit (64 Bit)
- Processor Unique Math Libraries: MMX, SSE, SSE2, 3DNow
- OpenEaagles developed using Microsoft Visual Studio
  – Tested on Windows, Linux (and Mac)

Ground Rules

- "C++ Coding Standards for OpenEaagles Project"
  - Strict coding practices (e.g., no public member variables)
  - Style consistent within a file (e.g., location of curly brackets)

- Real-Time rules (for time critical code)
  - Code must not wait for an external event that it has no control over (e.g., completion of I/O or network message, timers, semaphores)

Input Language

- Push method of argument parsing
  - Parser sends (pushes) arguments (slot pairs) to the object
  - Object responsible for checking and storing arguments
- Very simple, scheme-like language
  - Innate object-oriented support
- Parser generated using GNU’s Bison and Flex
  - Can be used to parse multiple files
- Easy to Parse
  - Single pass parser
  - Fast: Timing test: ~40K lines loaded in 1.5 seconds
- An off-the-shelf Scheme interpreter (e.g., siod, ksi) could be added as a scripting language, if needed.
- Easy to output from an object (serialize)

Input Language (cont)

- (classname <argumentList>)
  - Standard “form”, which creates an object of type ‘classname’ and applies the <argumentList> to the object
  - <argumentList> is a list of slot pairs that are passed to the new object
  - Slot pairs consist of an identifier with a colon and an object
    - “color: red”
    - “maxSpeed: 800”
  - Examples of forms:
    - (rgb red: 1.0 green: 0.5 blue: 0.5)
    - (MySystem num: 3 msg: “test this”)
  - When a slot name is not given, it’s position in the argument list (index) is used as the slot name
    - Example: (rgb 1.0 0.5 0.5) becomes (rgb 1: 1.0 2: 0.5 3: 0.5)

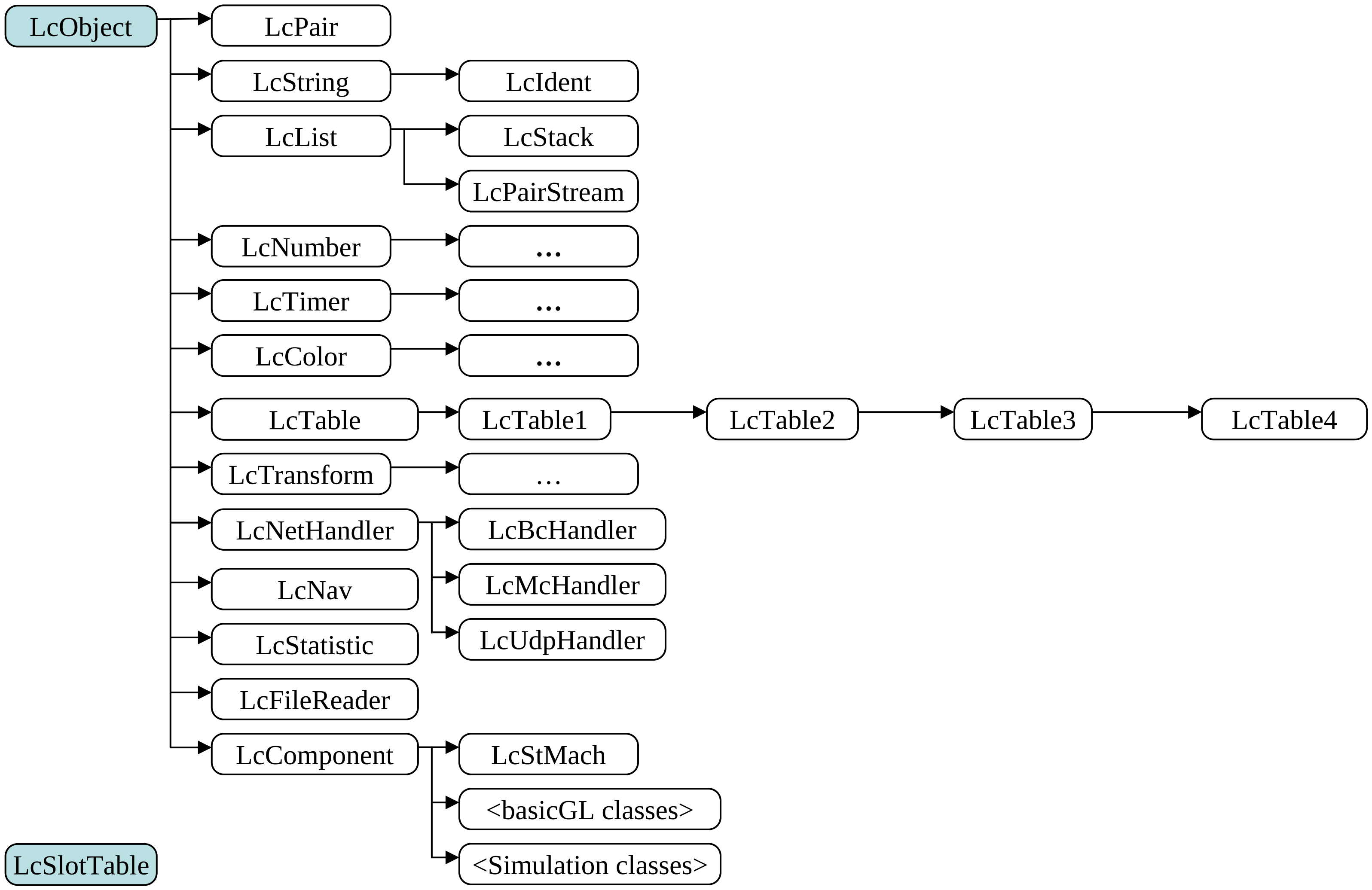
Input Language (cont)

- Types of objects on slot pairs
  - Real numbers
    - temperature: 98.6
  - Integer numbers
    - loops: 5
  - Booleans: values are ‘true’ and ‘false’
    - visible: true
  - Identifiers – single words
    - color: red
  - Strings – Text within double quotes
    - message: “fee fie foe foo”
  - Vectors – Lists of numbers inside square brackets
    - values: [15 0 4 12 -5 1]
  - Forms
    - color: (rgb   0.0   0.0   1.0)
  - List of slot pairs – slot pairs within curly brackets
    - list: { temperature: 98.6   loops: 5   visible: true   message: “Hello World!” }

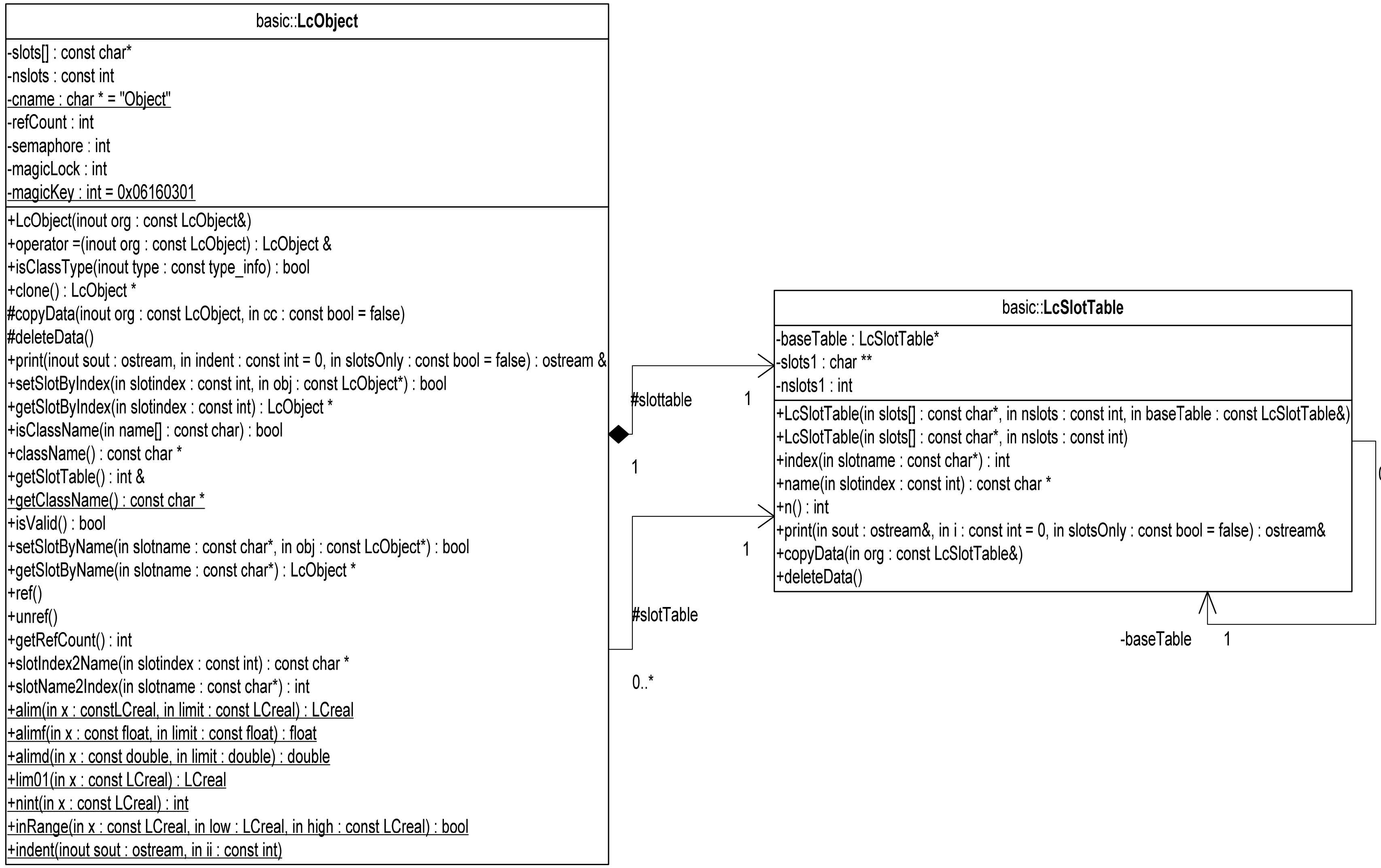
Input Language (cont)

```
// Simple Example
( MfdPage
    components: {
        // Green worm object
        worm1: ( Worm
            selectName: 111
            color: green
            speed: 10
            startAngle: ( Degrees 30 )
        )
        // Azimuth Pointer
        ptr1: ( AzPtr
           selectName: 281
           transform: ( Translation 0.0 -0.03 )
           components: {
               ( Line color: blue vertices: { [ 0 0 ] [ 0 -0.1 ] } )
               ( SpecialText
                   transform: { ( Translation -0.05 -0.3 ) ( Scale 0.1 ) }
                   text: "AZ"
               )
           }
        )
    }
)
```

Basic - Class Hierarchy



LcObject Static Structure


basic::**LcObject**
-slots[] : const char*
-nslots : const int
-cname : char * = "Object"
-refCount : int
-semaphore : int
-magicLock : int
-magicKey : int = 0x06160301
+LcObject(inout org : const LcObject&)
+operator =(inout org : const LcObject) : LcObject &
+isClassType(inout type : const type_info) : bool
+clone() : LcObject *
#copyData(inout org : const LcObject, in cc : const bool = false)
#deleteData()
+print(inout sout : ostream, in indent : const int = 0, in slotsOnly : const bool = false) : ostream &
+setSlotByIndex(in slotindex : const int, in obj : const LcObject*) : bool
+getSlotByIndex(in slotindex : const int) : LcObject *
+isClassName(in name[] : const char) : bool
+className() : const char *
+getSlotTable() : int &
+getClassName() : const char *
+isValid() : bool
+setSlotByName(in slotname : const char*, in obj : const LcObject*) : bool
+getSlotByName(in slotname : const char*) : LcObject *
+ref()
+unref()
+getRefCount() : int
+slotIndex2Name(in slotindex : const int) : const char *
+slotName2Index(in slotname : const char*) : int
+alim(in x : constLCreal, in limit : const LCreal) : LCreal
+alimf(in x : const float, in limit : const float) : float
+alimd(in x : const double, in limit : double) : double
+lim01(in x : const LCreal) : LCreal
+nint(in x : const LCreal) : int
+inRange(in x : const LCreal, in low : LCreal, in high : const LCreal) : bool
+indent(inout sout : ostream, in ii : const int)
#slottable
1
1
#slotTable
1
0..*
basic::**LcSlotTable**
-baseTable : LcSlotTable*
-slots1 : char **
-nslots1 : int
+LcSlotTable(in slots[] : const char*, in nslots : const int, in baseTable : const LcSlotTable&)
+LcSlotTable(in slots[] : const char*, in nslots : const int)
+index(in slotname : const char*) : int
+name(in slotindex : const int) : const char *
+n() : int
+print(in sout : ostream&, in i : const int = 0, in slotsOnly : const bool = false) : ostream&
+copyData(in org : const LcSlotTable&)
+deleteData()
0..*
-baseTable
1

Basic – LcObject

- Generic class that provides a common type for managing and passing objects, class type, name information, simple garbage collection, copy functions and support for an input (configuration) file parser

- Our “System” object – since C++ doesn’t provide one

- The "Lc" prefix stands for "Link class" in honor of Ed Link, who invented the "Link Blue Box", which was the first flight simulator. This will be removed in upcoming releases as namespaces are now being used.

Basic – LcObject (cont)

- Class Type – (type_info class)
  - typeid(expression)
    - Returns reference to the type_info class
    - Standard C++
  - bool isClassType(type_info& T)
    - Returns true when the object is class type T or derived from class type T
  - dynamic_cast<T*> (&object)
    - Standard C++

- Class Name
  - const char* className()
    - Member function that returns the object’s class name
    - This name is used in the configuration files and is not the class name from type_info.
  - const char* getClassName()
    - Static member function that returns the name of the class.
  - bool isClassName(const char* name)
    - Returns true when the object is of class type ‘name’ or derived from class type ‘name’

Basic – LcObject (cont)

- Copy functions: copy constructor, copy operator and clone()
  - All three copy functions are required and are defined by macros
    - MyClass(MyClass& source) /* Copy constructor */
    - MyClass& operator=(MyClass& source) /* Copy operator */
    - LcObject* clone() /* Clone function */

- copyData(MyClass& source, bool ccFlag)
  - Required user defined function
  - Copies all necessary data from the source object
  - ccFlag is true if called from the copy constructor; therefore this object's member data must be initialized by copyData()
  - Must copy the base class' data first!
    - BaseClass::copyData(source)

- deleteData()
  - Required user defined function
  - Disposes of (or unref()) all data owned/managed by this object
  - Can be called multiple times, so data needs to be disposed of cleanly
    - (e.g., test for pointer not equal to zero before unref() an object and setting the pointer to zero after the object is unref()'d)
  - Must delete the base class data last!
    - BaseClass::deleteData()

Basic – LcObject (cont)

- Garbage collection (simple)
  - ref()
    - Increments the reference count
    - Throws LcExpInvalidRefCount if the reference count is invalid
  - unref()
    - Decrements the reference count
    - Deletes this object when the reference count reaches zero
  - getRefCount()
    - Returns the current value of the reference count
  - The LcObject constructor and the clone() function initialize the reference count to one (i.e., pre-referenced by the creator)
  - SPtr<T>
    - “Smart Pointer” Template
    - Contains a pointer to type T, which must be derived from LcObject
    - Handles the ref() and unref() of objects of type T.

Basic – LcObject (cont)

- Exceptions
  - LcObject::LcException
    - All exceptions derived from this type
  - Throw a pointer to the exception; not the exception!
  - Plan to use the exception from the Standard Template Library

- bool isValid()
  - Returns true if the object is valid
  - Magic number match
  - Reference count is greater than zero
  - Virtual function: derived classes can add additional validity checks

Basic – support.h

- Included by Object.h
- Defines common types, templates, functions and constants
  - Header files: <osg/Math>, <iostream>, <typeinfo> and <string>
    - WIN32 only: <windows.h>
    - Linux: <values.h>
  - Type “LCreal” as double or float
  - LCreal math functions (e.g., lcSqrt(), lcSin(), lcCos(), etc.)
  - Standard event message tokens
  - Smart pointer template, SPtr, which handles ref() and unref()
  - Quick queue and stack templates: QQueue and QStack
    - Fast; Single reader/writer; Single thread

Basic – macros.h

- Macros define standardized (boilerplate) code
- Included by Object.h
- DECLARE_SUBCLASS(ThisType,BaseType)
  - Used in the class specification (*.h) to declare all standard member functions and data
  - Defines type ‘BaseClass’, which is used to reference the base class members without specifying the actual base class by name
- IMPLEMENT_SUBCLASS(ThisType,”ClassName”)
  - Used in the class body (*.cpp) to define all standard member functions and data
- Example:
  - Header File: Radar.h

```
class Radar : public Sensor {
    DECLARE_SUBCLASS(Radar,Sensor)
    /* class Radar code here */
}
```

  - Source File: Radar.cpp

```
#include “Radar.h”
    IMPLEMENT_SUBCLASS(Radar,”Radar”)
    /* Radar code here */
```

Basic – LcSlotTable

- Purpose: Defines the names of slots accepted by the class and maps these slot names to slot indexes
- Contains a pointer to the base class' slot table
  - Slot names that are not found in a class' slot table are passed up to the base class
  - Slot names are inherited by derived classes
  - Derived classes can override base class slot names
- Slot tables are defined by macros in the class' source *.cpp file
- Class constructors must include the SET_SLOTTABLE macro
- Slot tables are defined by BEGIN_SLOTTABLE() and END_SLOTTABLE()

```
BEGIN_SLOTTABLE(MyClass)
    "slotA",        /* index #1 */
    "slotB",        /* index #2 */
    "slotC",        /* index #3 */
END_SLOTTABLE(MyClass)
```

Basic – LcObject (cont)

- LcSlotTable & getSlotTable()
  - Static member function that returns a reference to the class' slot table

- Slot functions
  - bool setSlotByName(char* name, LcObject* arg)
  - bool setSlotByIndex(int index, LcObject* arg)
    - Parser and other classes use setSlotByName()
    - Derived classes implement setSlotByIndex()
    - True is returned if slot was set to argument and false is returned if there was an error

  - LcObject* getSlotByName(char* name)
  - LcObject* getSlotByIndex(int index)
    - Returns an object containing the value of the slot

  - int slotName2Index(char* name)
  - const char* slotIndex2Name(int index)

- ostream& print(ostream& sout, int indent = 0, bool slotsOnly = false)
  - Prints object's name and slots to the ostream
  - If the 'slotsOnly' argument is true, only the slots are printed
  - Used to save current configuration or data collection

Basic – LcObject (cont)

- setSlotByIndex() can be defined using macros

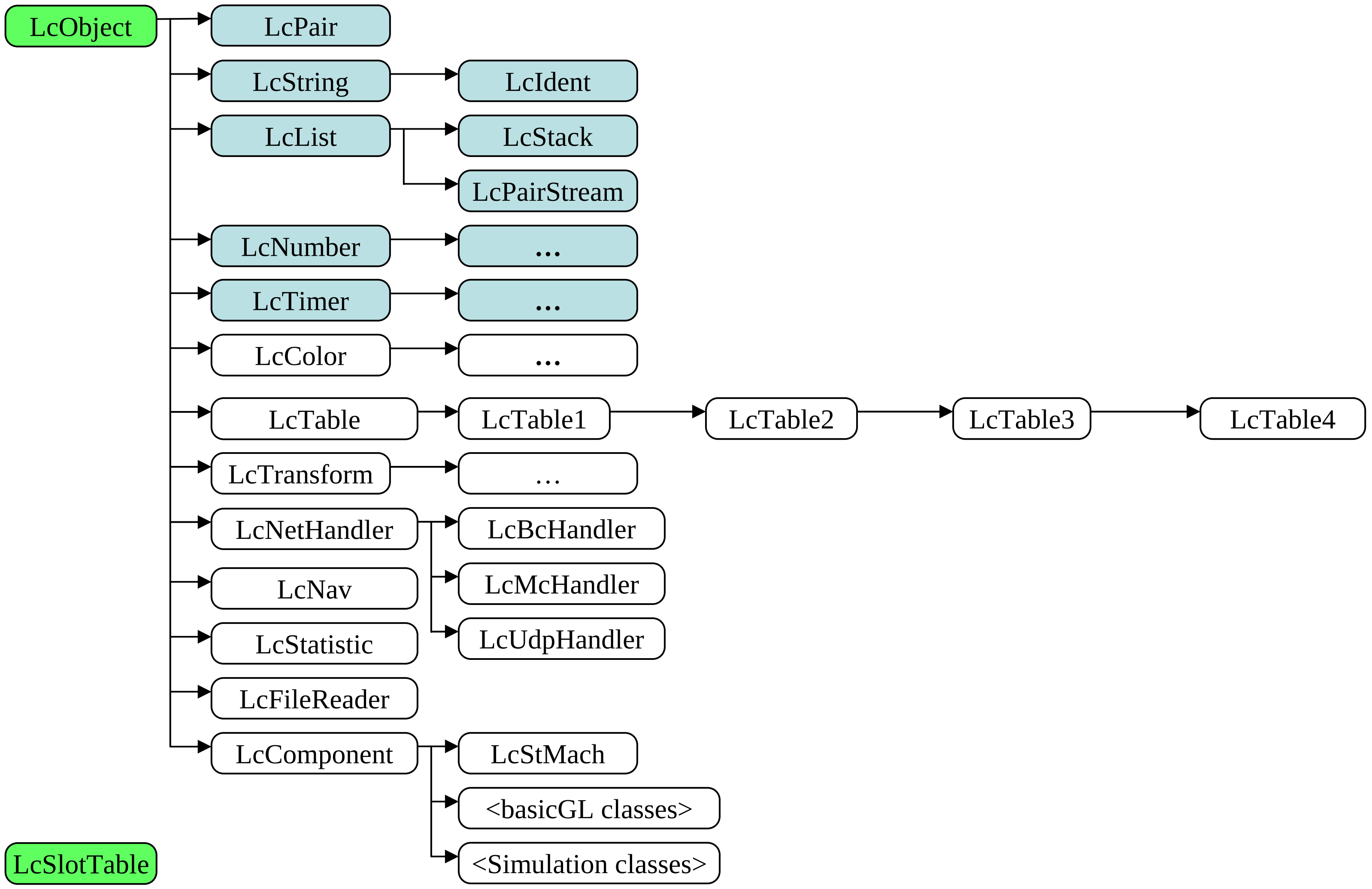
- Example:

```
BEGIN_SLOT_MAP(MyClass)
  ON_SLOT(1,setSlotA,LcNumber)
  ON_SLOT(2,setSlotB,LcString)
  ON_SLOT(2,setSlotB,LcList)
  ON_SLOT(3,setSlotC,LcIdent)
END_SLOT_MAP()
```

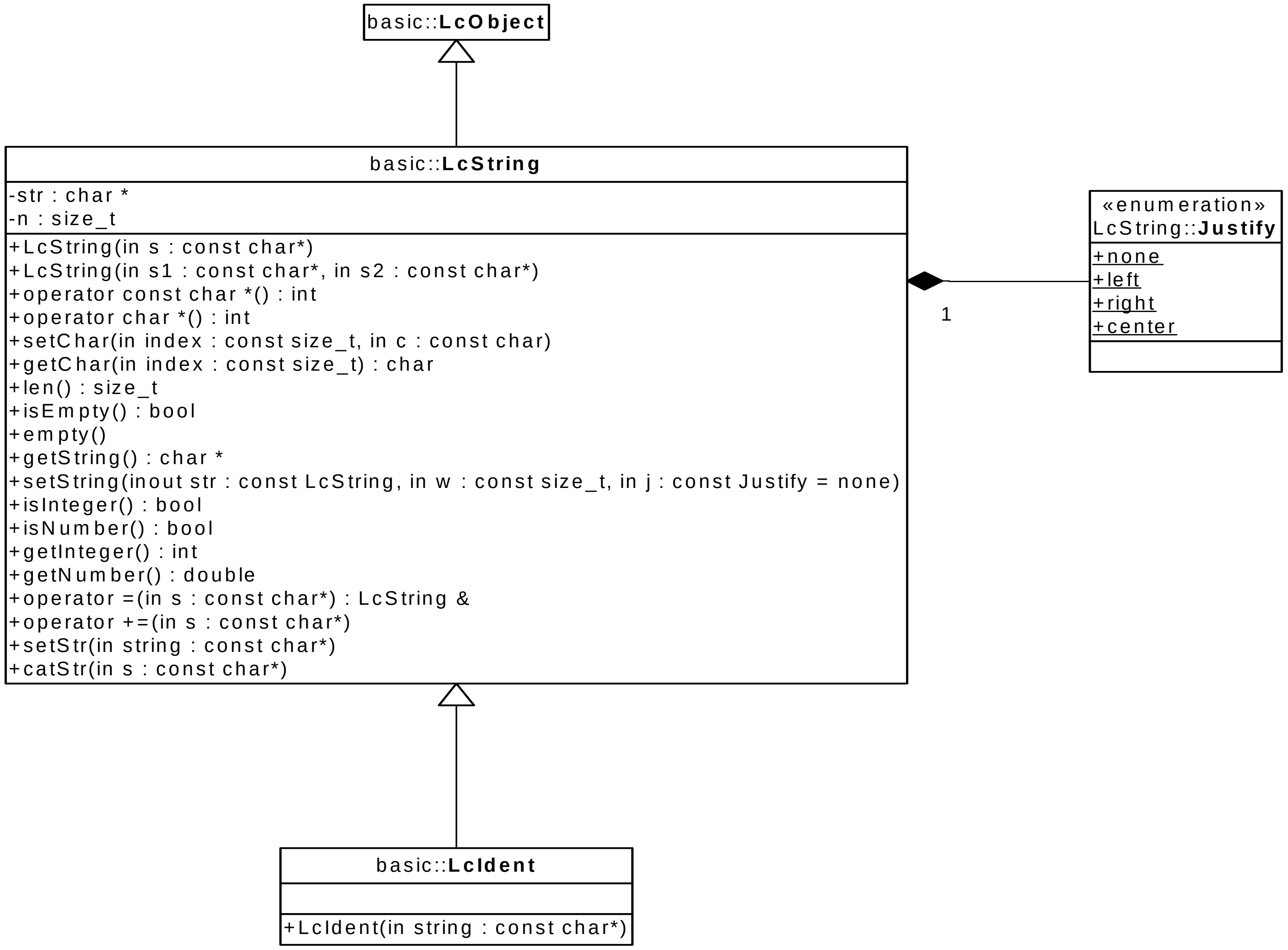
- BEGIN_SLOT_MAP() and END_SLOT_MAP() define the start and end of the map
- ON_SLOT() maps an index and object type to a function to handle the slot
  - First argument is the slot index to match
  - Second argument is the appropriate 'set' function to call if the object's type matches the third argument

- Set functions are implemented for each class
  - Standard set function template: bool setXXX(T* obj)
  - Each function must error check the object
  - Return true if the slot was set and false if there was an error

Basic - Class Hierarchy



LcString Static Structure


basic::**LcObject**
basic::**LcString**
-str : char *
-n : size_t
+LcString(in s : const char*)
+LcString(in s1 : const char*, in s2 : const char*)
+operator const char *() : int
+operator char *() : int
+setChar(in index : const size_t, in c : const char)
+getChar(in index : const size_t) : char
+len() : size_t
+isEmpty() : bool
+empty()
+getString() : char *
+setString(inout str : const LcString, in w : const size_t, in j : const Justify = none)
+isInteger() : bool
+isNumber() : bool
+getInteger() : int
+getNumber() : double
+operator =(in s : const char*) : LcString &
+operator +=(in s : const char*)
+setStr(in string : const char*)
+catStr(in s : const char*)
1
«enumeration»
LcString::**Justify**
+none
+left
+right
+center
basic::**LcIdent**
+LcIdent(in string : const char*)

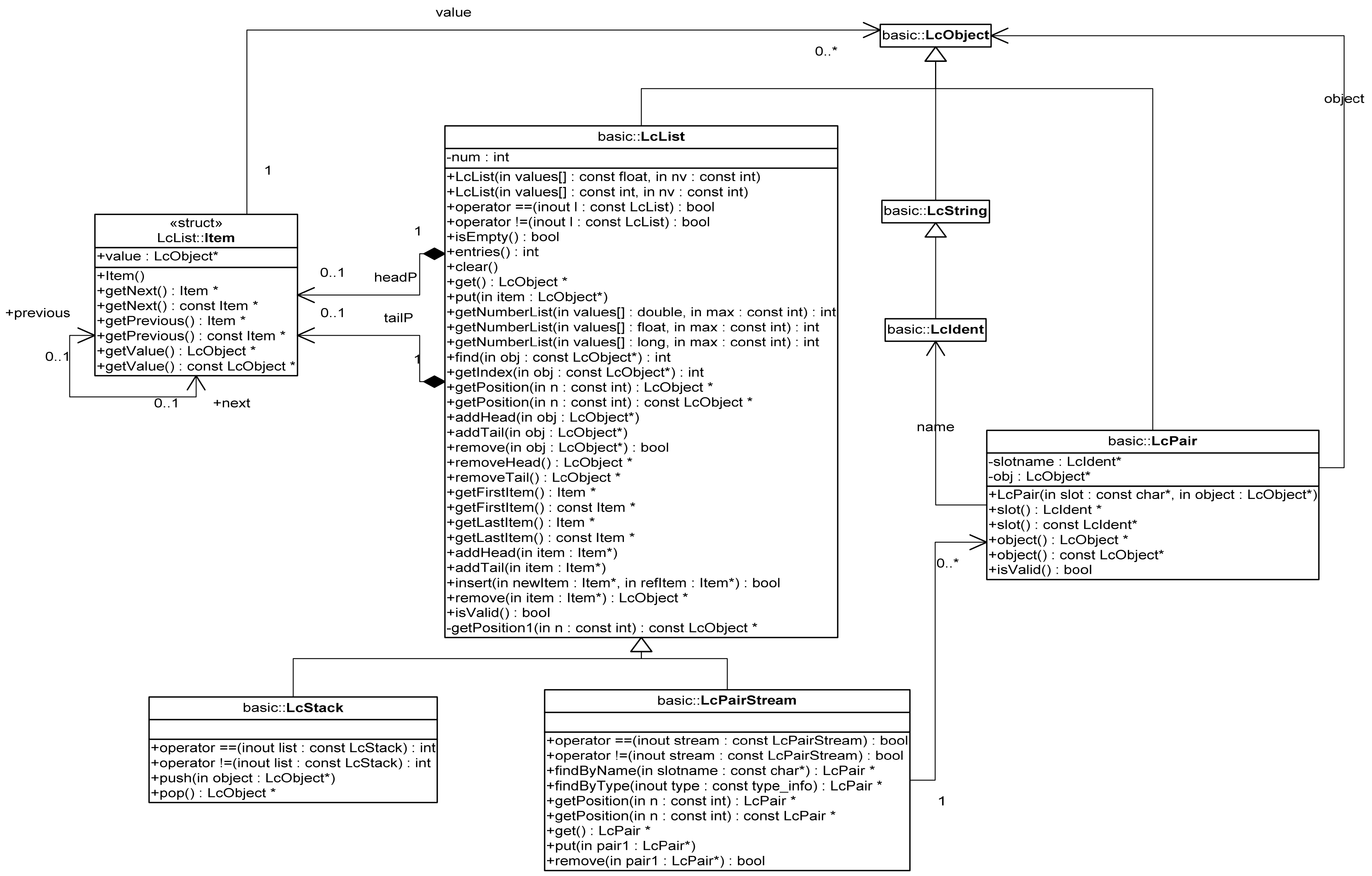
Basic – LcString

- Purpose: Container for character strings
- Initial character string and length are set by the constructors
- Member functions (subset)
  - char* getString() /* returns a copy of the string */
  - int len() /* returns the length of the string */
  - char getChar(int n) /* returns the n’th character */
  - bool isNumber() /* returns true if number “123.45” */
  - float getNumber() /* returns the value of the number */
- Class LcIdent
  - Derived from LcString
  - Used for single word identifiers

Basic – LcString (cont)

- Supported Operators
  - Assignment Operators: =
  - Concatenation Operators: + +=
  - Comparison Operators: < <= == >= > !=
  - iostream Operations: << >>
- setString(LcString& str2, int length, Justify jvalue)
  - The LcString is set to the character string in ‘str2’ with a new length and positioned according to the justification parameter
  - Justify: left, right, center and none
- Input File
  - LcString: “text strings within quotes”
  - LcIdent: single words without quotes and slot names

LcList Static Structure


value
0..*
basic::LcObject
object
1
basic::LcList
-num : int
+LcList(in values[] : const float, in nv : const int)
+LcList(in values[] : const int, in nv : const int)
+operator ==(inout l : const LcList) : bool
+operator !=(inout l : const LcList) : bool
+isEmpty() : bool
+entries() : int
+clear()
+get() : LcObject *
+put(in item : LcObject*)
+getNumberList(in values[] : double, in max : const int) : int
+getNumberList(in values[] : float, in max : const int) : int
+getNumberList(in values[] : long, in max : const int) : int
+find(in obj : const LcObject*) : int
+getIndex(in obj : const LcObject*) : int
+getPosition(in n : const int) : LcObject *
+getPosition(in n : const int) : const LcObject *
+addHead(in obj : LcObject*)
+addTail(in obj : LcObject*)
+remove(in obj : LcObject*) : bool
+removeHead() : LcObject *
+removeTail() : LcObject *
+getFirstItem() : Item *
+getFirstItem() : const Item *
+getLastItem() : Item *
+getLastItem() : const Item *
+addHead(in item : Item*)
+addTail(in item : Item*)
+insert(in newItem : Item*, in refItem : Item*) : bool
+remove(in item : Item*) : LcObject *
+isValid() : bool
-getPosition1(in n : const int) : const LcObject *
basic::LcString
basic::LcIdent
«struct»
LcList::Item
+value : LcObject*
+Item()
+getNext() : Item *
+getNext() : const Item *
+getPrevious() : Item *
+getPrevious() : const Item *
+getValue() : LcObject *
+getValue() : const LcObject *
1
0..1
headP
0..1
tailP
1
+previous
0..1
0..1
+next
name
basic::LcPair
-slotname : LcIdent*
-obj : LcObject*
+LcPair(in slot : const char*, in object : LcObject*)
+slot() : LcIdent *
+slot() : const LcIdent*
+object() : LcObject *
+object() : const LcObject*
+isValid() : bool
0..*
basic::LcStack
+operator ==(inout list : const LcStack) : int
+operator !=(inout list : const LcStack) : int
+push(in object : LcObject*)
+pop() : LcObject *
basic::LcPairStream
+operator ==(inout stream : const LcPairStream) : bool
+operator !=(inout stream : const LcPairStream) : bool
+findByName(in slotname : const char*) : LcPair *
+findByType(inout type : const type_info) : LcPair *
+getPosition(in n : const int) : LcPair *
+getPosition(in n : const int) : const LcPair *
+get() : LcPair *
+put(in pair1 : LcPair*)
+remove(in pair1 : LcPair*) : bool
1

Basic – LcList

- Purpose: Maintains a list of objects (LcObject)

- Member functions
  - put(LcObject* obj)
    - Puts an object on the end of the list and references the object, ref().
  - LcObject* get()
    - Removes the first object on the list and returns a pointer to the object. Empty lists will return null(0). Ownership of the object is passed to the caller (i.e., this routine does not unref() the object).
  - bool remove(LcObject* obj)
    - Removes an object from the list. Returns true if the object was found and removed, otherwise false is returned. The object is unref(), and therefore, possibly deleted.
  - int entries()
    - Returns the number of objects in the list

- Number list
  - LcList(float values[], int n)
    - Special constructor that creates a list of LcNumber classes containing the values from the array
  - int getNumberList(float values[], int max)
    - Fills the array, values[], with up to ‘max’ values found in the list
    - Returns the actual number of values found
  - Input file: [10 20 30]

Basic – LcList (cont)

- LcList::Item class
  - Represents a node on the list
  - Pointer to the object
  - Pointers to the next and previous list items
  - LcList::Item member functions
    - LcObject* getValue() /* returns a pointer to the object */
    - Item* getNext() /* returns a pointer to the next item in the list */
    - Item* getPrevious() /* returns a pointer to the previous item */

- More LcList member functions
  - Item* getFirstItem()
    - Returns a pointer to the first Item on the list
  - Item* getLastItem()
    - Returns a pointer to the last item on the list

- Example:

```
LcList* list = new LcList();
// … code to add objects to list …
For (LcList::Item* item = list->getFirstItem(); item != 0; item = item->getNext()) {
      LcObject* obj = item->getValue()
      // … code to use the object…
}
```

Basic – LcPair

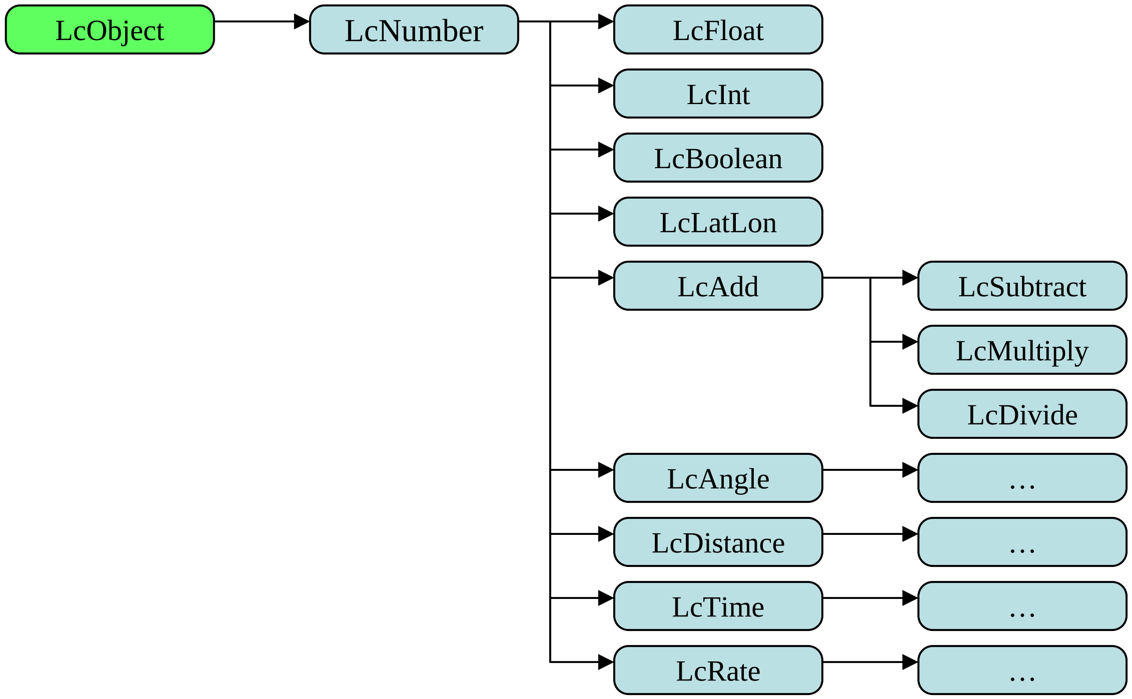
- LcPair: Container for name/object pairs (slot pairs)
  - Pair set by constructor only
    - LcPair(“Blue1”, &f22Player)
  - Member functions
    - LcIdent* slot()   /* Returns a pointer to the name */
    - LcObject* object()   /* Returns a pointer to the object */
  - Input file: Slot Pairs

    name: object
- LcPairStream: list of LcPair objects
  - Derived from LcList
  - Input file: general lists and argument lists in forms

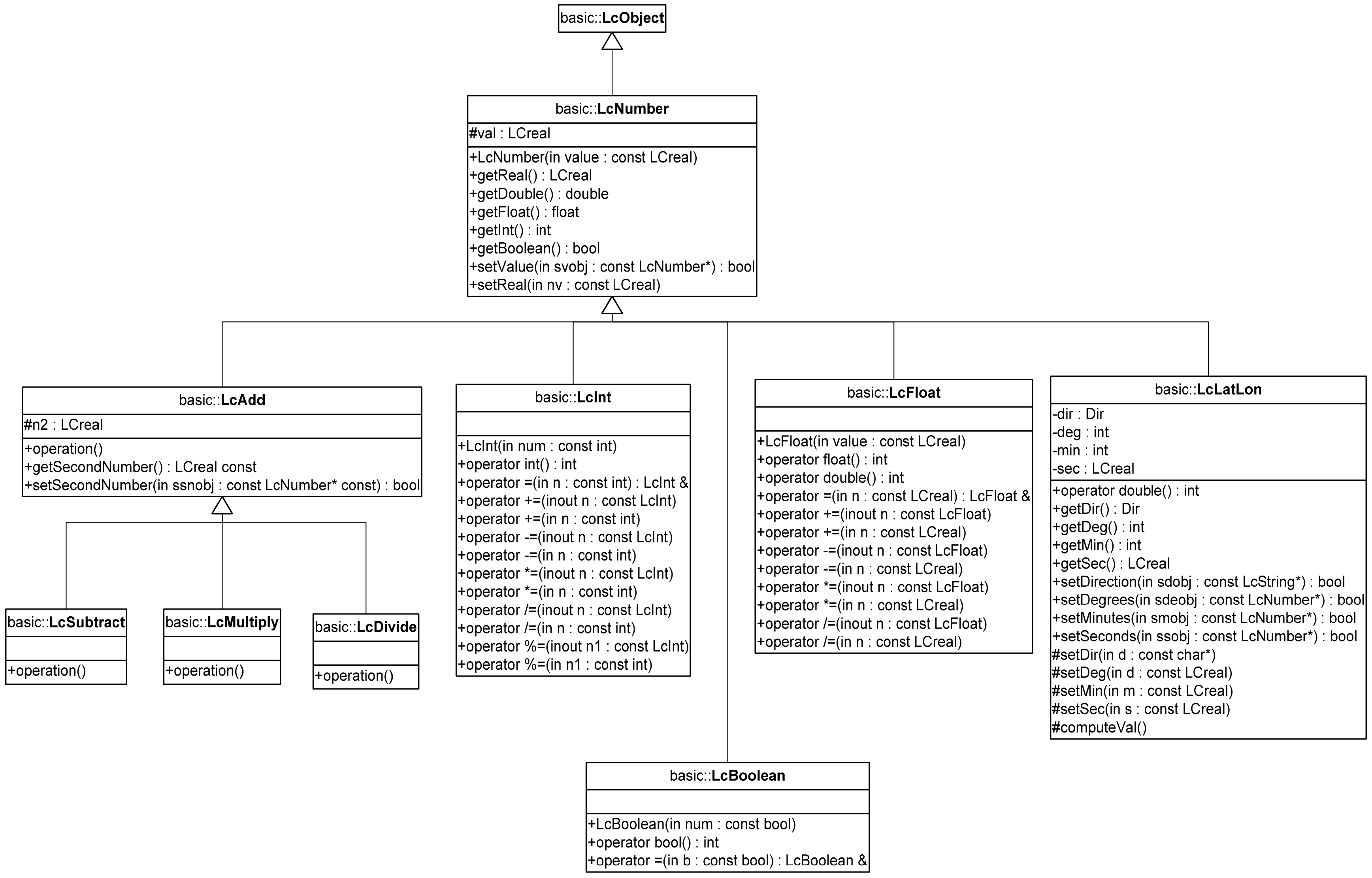
    { n1: obj1   n2: obj2   n3: obj3 }

    (ClassName   slot1: obj1 slot2: obj2 slot3: obj3)

Basic – LcNumber class hierarchy



LcNumber Static Structure


basic::LcObject
basic::LcNumber
#val : LCreal
+LcNumber(in value : const LCreal)
+getReal() : LCreal
+getDouble() : double
+getFloat() : float
+getInt() : int
+getBoolean() : bool
+setValue(in svobj : const LcNumber*) : bool
+setReal(in nv : const LCreal)
basic::LcAdd
#n2 : LCreal
+operation()
+getSecondNumber() : LCreal const
+setSecondNumber(in ssnobj : const LcNumber* const) : bool
basic::LcSubtract
+operation()
basic::LcMultiply
+operation()
basic::LcDivide
+operation()
basic::LcInt
+LcInt(in num : const int)
+operator int() : int
+operator =(in n : const int) : LcInt &
+operator +=(inout n : const LcInt)
+operator +=(in n : const int)
+operator -=(inout n : const LcInt)
+operator -=(in n : const int)
+operator *=(inout n : const LcInt)
+operator *=(in n : const int)
+operator /=(inout n : const LcInt)
+operator /=(in n : const int)
+operator %=(inout n1 : const LcInt)
+operator %=(in n1 : const int)
basic::LcFloat
+LcFloat(in value : const LCreal)
+operator float() : int
+operator double() : int
+operator =(in n : const LCreal) : LcFloat &
+operator +=(inout n : const LcFloat)
+operator +=(in n : const LCreal)
+operator -=(inout n : const LcFloat)
+operator -=(in n : const LCreal)
+operator *=(inout n : const LcFloat)
+operator *=(in n : const LCreal)
+operator /=(inout n : const LcFloat)
+operator /=(in n : const LCreal)
basic::LcLatLon
-dir : Dir
-deg : int
-min : int
-sec : LCreal
+operator double() : int
+getDir() : Dir
+getDeg() : int
+getMin() : int
+getSec() : LCreal
+setDirection(in sdobj : const LcString*) : bool
+setDegrees(in sdeobj : const LcNumber*) : bool
+setMinutes(in smobj : const LcNumber*) : bool
+setSeconds(in ssobj : const LcNumber*) : bool
#setDir(in d : const char*)
#setDeg(in d : const LCreal)
#setMin(in m : const LCreal)
#setSec(in s : const LCreal)
#computeVal()
basic::LcBoolean
+LcBoolean(in num : const bool)
+operator bool() : int
+operator =(in b : const bool) : LcBoolean &

Basic – LcNumber

- Purpose: Abstract containers for numbers

- The number is stored internally as type LCreal

- Access member functions
  - LCreal getReal()
  - double getDouble()
  - float getFloat()
  - int getInt()
  - bool getBoolean()

- Value set by the constructors, assignment operators or the set functions
  - setReal(LCreal number)
  - setValue(LcNumber* number)

Basic – LcNumber (cont)

- Derived classes
  - LcFloat
  - LcInt
  - LcBool
  - LcLatLong
  - Various units (Distance, time, angles, rates)

- Most derived classes support the following operators
  - Assignment operators: = += -= *= /=
  - Binary operators: + - * /
  - Comparison operators: == != < <= > >=
  - Input/Output stream operators: >> <<

- Input File
  - LcInt for integers
  - LcFloat for real numbers
  - LcBool for ‘true’ and ‘false’

Basic – Units

- Purpose: Containers for numbers with units

- Major unit classes are derived from LcNumber
- Minor unit classes derived from the major units
  - Example: LcDegrees and LcRadians are derived from LcAngle

- Major unit classes (examples)
  - LcAngle
    - Degrees, radians, semicircles
  - LcTime
    - Hours, minutes, seconds, milliseconds
  - LcDistance
    - Kilometers, meters, centimeters, nautical miles, statute miles, feet, inches
  - LcVelocity
    - Ratio of two units (LcDistance & LcTime)

- Operators
  - Binary operators: + - * /
  - Comparison operators: == != < <= > >=

- Handy conversion constants (e.g., feet to meters, degrees to radians, etc.)’

- Handy limit functions (e.g., angles limited to +/- 180 degrees)

Basic – Units (cont)

- Input file examples
  - (Degrees 12)
  - (Feet 12345)
  - (Velocity 678 (Feet) (Second))

- All major unit classes provide conversion routines; examples …
  - LcAngle::degreesToRadians()
  - LcDistance::feetToMeters()
  - LcTime::hoursToMinutes()

- set(LCreal source)
  - Sets the object's value to source without conversion

- set(MajorUnitType& source)
  - Sets the object's value to source with conversion

- LCreal convert(MajorUnitType& source)
  - Returns a value converted from the source

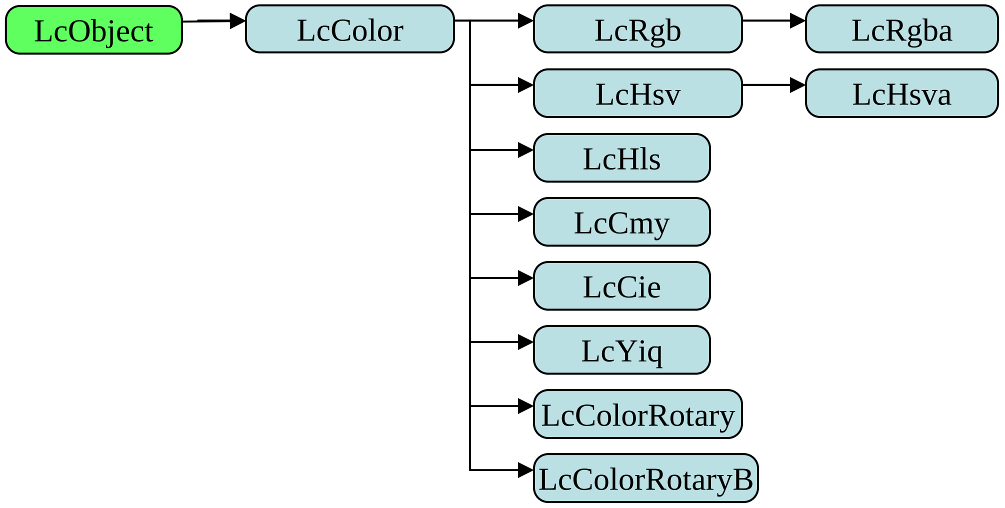
Basic – Units (cont)

```
// ---
// Example 1
// ---
LcDegrees d(45.0);  // Our LcDegrees variable
LcRadian r(1.5);    // Our LcRadian variable

// ‘v1’ is the value of ‘r’ converted to degrees.
// The value of ‘d’ is unchanged.
double v1 = d.convert(r);

// Set the value of ‘d’ to ‘r’ with conversion
d.set(r);

// ---
// Example 2
// ---
void func1(LcAngle& angle)
{
    LcDegree d;
    double value = d.convert(angle);
    // … code to use the angle as degrees, value …
}
```

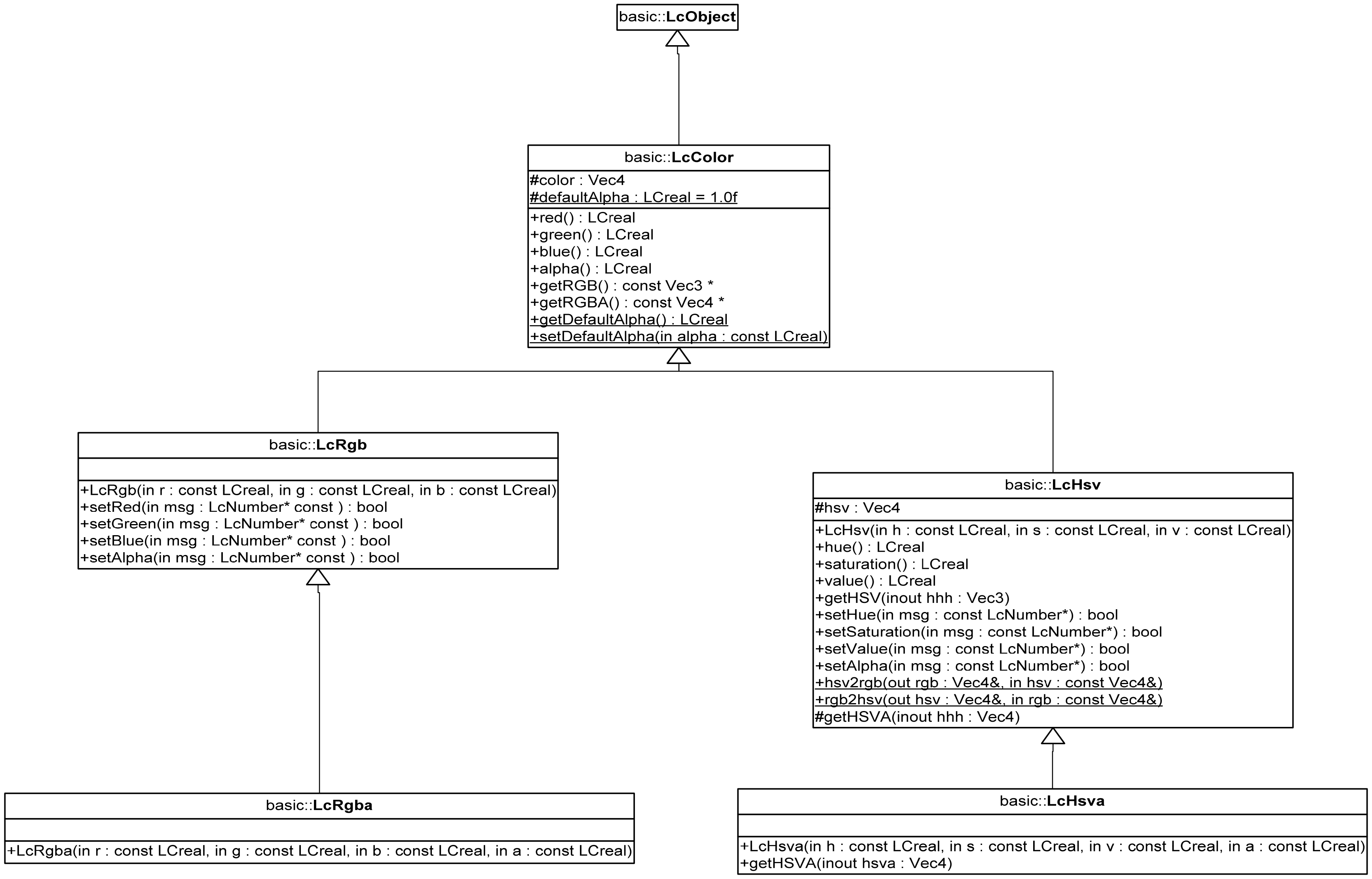
Basic – LcTimer

- Purpose: General purpose up/down timers
- Derived classes: LcUpTimer, LcDownTimer
- The timer's direction and default reset value are set by the constructors
- Member functions
  - float time() Returns time current value of the timer
  - start() Starts the timer
  - stop() Stops the timer
  - reset()
  - reset(float rTime)
    - Timer reset functions
  - bool alarm()
  - bool alarm(float alarmTime)
    - Timer alarm functions: return true when the timer reaches the alarm time
- Static functions (all timers)
  - updateTimers(float deltaTime) Updates all timers
  - freeze(bool frzFlag) Starts or stops all times

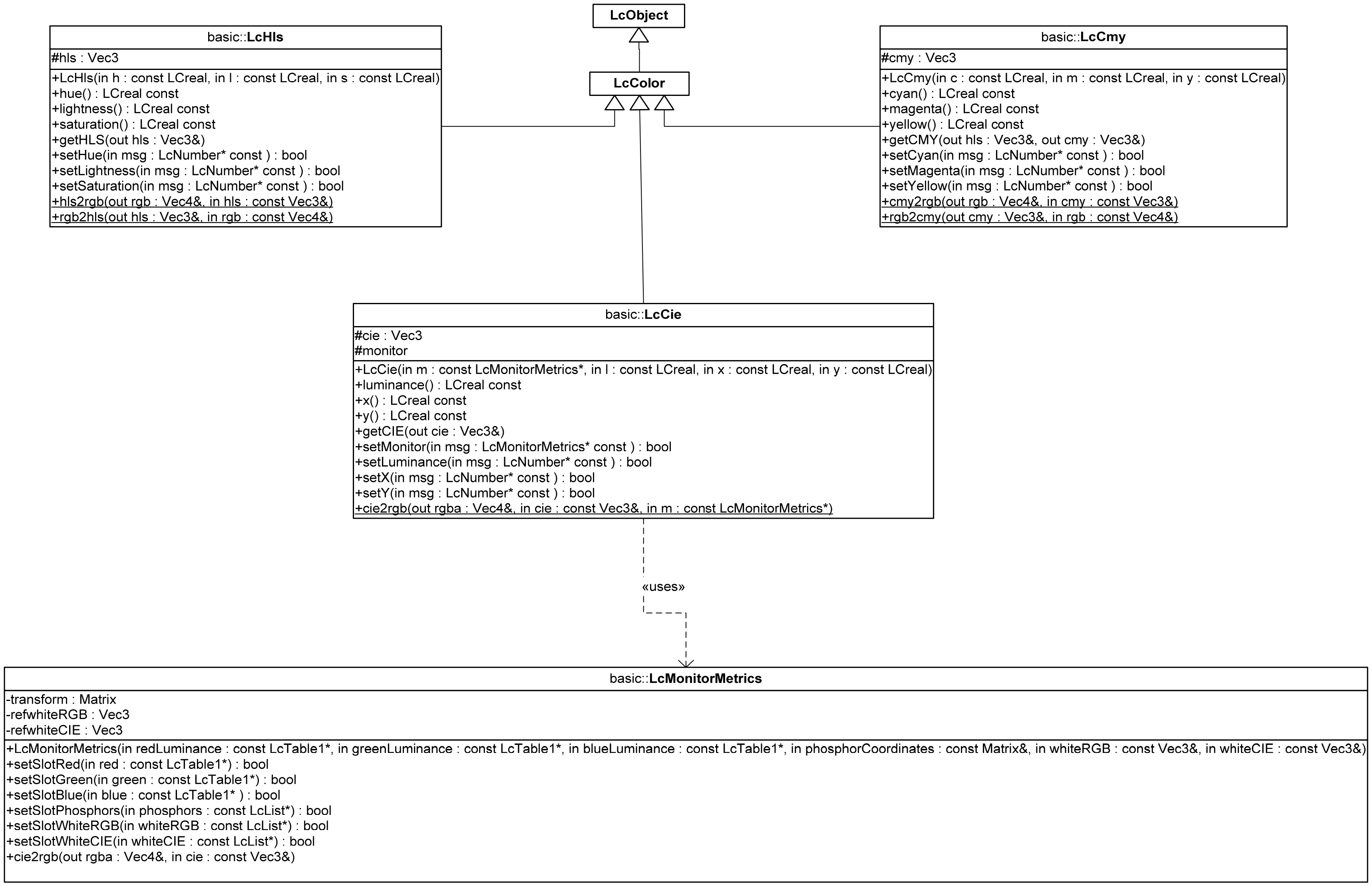
Basic - Class Hierarchy



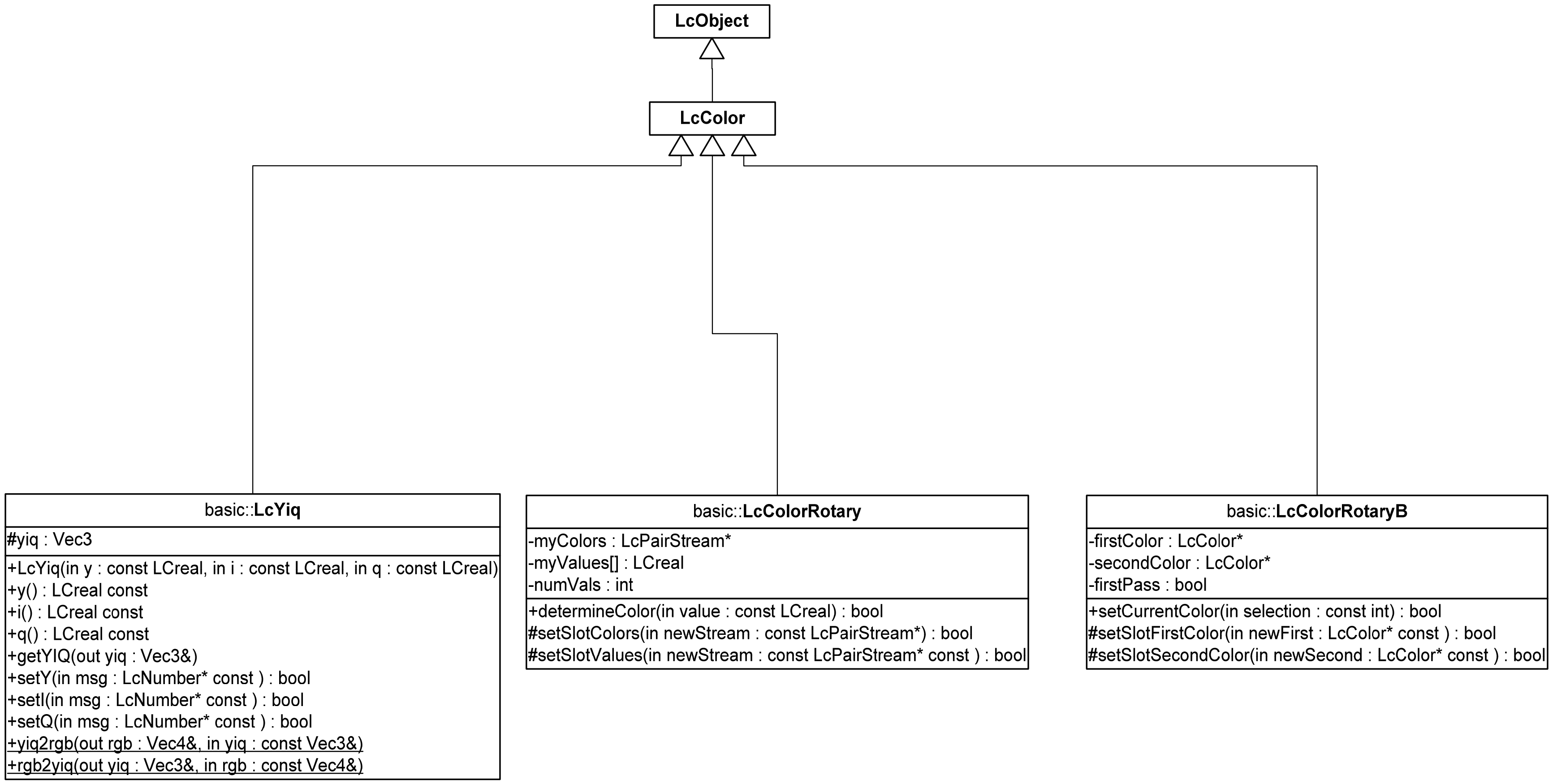
LcColor Static Structure


basic::LcObject
basic::LcColor
#color : Vec4
#defaultAlpha : LCreal = 1.0f
+red() : LCreal
+green() : LCreal
+blue() : LCreal
+alpha() : LCreal
+getRGB() : const Vec3 *
+getRGBA() : const Vec4 *
+getDefaultAlpha() : LCreal
+setDefaultAlpha(in alpha : const LCreal)
basic::LcRgb
+LcRgb(in r : const LCreal, in g : const LCreal, in b : const LCreal)
+setRed(in msg : LcNumber* const) : bool
+setGreen(in msg : LcNumber* const) : bool
+setBlue(in msg : LcNumber* const) : bool
+setAlpha(in msg : LcNumber* const) : bool
basic::LcHsv
#hsv : Vec4
+LcHsv(in h : const LCreal, in s : const LCreal, in v : const LCreal)
+hue() : LCreal
+saturation() : LCreal
+value() : LCreal
+getHSV(inout hhh : Vec3)
+setHue(in msg : const LcNumber*) : bool
+setSaturation(in msg : const LcNumber*) : bool
+setValue(in msg : const LcNumber*) : bool
+setAlpha(in msg : const LcNumber*) : bool
+hsv2rgb(out rgb : Vec4&, in hsv : const Vec4&)
+rgb2hsv(out hsv : Vec4&, in rgb : const Vec4&)
#getHSVA(inout hhh : Vec4)
basic::LcRgba
+LcRgba(in r : const LCreal, in g : const LCreal, in b : const LCreal, in a : const LCreal)
basic::LcHsva
+LcHsva(in h : const LCreal, in s : const LCreal, in v : const LCreal, in a : const LCreal)
+getHSVA(inout hsva : Vec4)

LcColor Static Structure (cont)


LcObject
LcColor
basic::LcHls
#hls : Vec3
+LcHls(in h : const LCreal, in l : const LCreal, in s : const LCreal)
+hue() : LCreal const
+lightness() : LCreal const
+saturation() : LCreal const
+getHLS(out hls : Vec3&)
+setHue(in msg : LcNumber* const) : bool
+setLightness(in msg : LcNumber* const) : bool
+setSaturation(in msg : LcNumber* const) : bool
+hls2rgb(out rgb : Vec4&, in hls : const Vec3&)
+rgb2hls(out hls : Vec3&, in rgb : const Vec4&)
basic::LcCmy
#cmy : Vec3
+LcCmy(in c : const LCreal, in m : const LCreal, in y : const LCreal)
+cyan() : LCreal const
+magenta() : LCreal const
+yellow() : LCreal const
+getCMY(out hls : Vec3&, out cmy : Vec3&)
+setCyan(in msg : LcNumber* const) : bool
+setMagenta(in msg : LcNumber* const) : bool
+setYellow(in msg : LcNumber* const) : bool
+cmy2rgb(out rgb : Vec4&, in cmy : const Vec3&)
+rgb2cmy(out cmy : Vec3&, in rgb : const Vec4&)
basic::LcCie
#cie : Vec3
#monitor
+LcCie(in m : const LcMonitorMetrics*, in l : const LCreal, in x : const LCreal, in y : const LCreal)
+luminance() : LCreal const
+x() : LCreal const
+y() : LCreal const
+getCIE(out cie : Vec3&)
+setMonitor(in msg : LcMonitorMetrics* const) : bool
+setLuminance(in msg : LcNumber* const) : bool
+setX(in msg : LcNumber* const) : bool
+setY(in msg : LcNumber* const) : bool
+cie2rgb(out rgba : Vec4&, in cie : const Vec3&, in m : const LcMonitorMetrics*)
«uses»
basic::LcMonitorMetrics
-transform : Matrix
-refwhiteRGB : Vec3
-refwhiteCIE : Vec3
+LcMonitorMetrics(in redLuminance : const LcTable1*, in greenLuminance : const LcTable1*, in blueLuminance : const LcTable1*, in phosphorCoordinates : const Matrix&, in whiteRGB : const Vec3&, in whiteCIE : const Vec3&)
+setSlotRed(in red : const LcTable1*) : bool
+setSlotGreen(in green : const LcTable1*) : bool
+setSlotBlue(in blue : const LcTable1*) : bool
+setSlotPhosphors(in phosphors : const LcList*) : bool
+setSlotWhiteRGB(in whiteRGB : const LcList*) : bool
+setSlotWhiteCIE(in whiteCIE : const LcList*) : bool
+cie2rgb(out rgba : Vec4&, in cie : const Vec3&)

LcColor Static Structure (cont)


LcObject
LcColor
basic::LcYiq
#yiq : Vec3
+LcYiq(in y : const LCreal, in i : const LCreal, in q : const LCreal)
+y() : LCreal const
+i() : LCreal const
+q() : LCreal const
+getYIQ(out yiq : Vec3&)
+setY(in msg : LcNumber* const) : bool
+setI(in msg : LcNumber* const) : bool
+setQ(in msg : LcNumber* const) : bool
+yiq2rgb(out rgb : Vec4&, in yiq : const Vec3&)
+rgb2yiq(out yiq : Vec3&, in rgb : const Vec4&)
basic::LcColorRotary
-myColors : LcPairStream*
-myValues[] : LCreal
-numVals : int
+determineColor(in value : const LCreal) : bool
#setSlotColors(in newStream : const LcPairStream*) : bool
#setSlotValues(in newStream : const LcPairStream* const) : bool
basic::LcColorRotaryB
-firstColor : LcColor*
-secondColor : LcColor*
-firstPass : bool
+setCurrentColor(in selection : const int) : bool
#setSlotFirstColor(in newFirst : LcColor* const) : bool
#setSlotSecondColor(in newSecond : LcColor* const) : bool

Basic – LcColor

- Purpose: Provides an RGBA color container

- LcColor member functions
  - LCreal getRed()
  - LCreal getBlue()
  - LCreal getGreen()
  - LCreal getAlpha()

  - osg::vec3 getRGB()
  - osg::vec4 getRGBA()

- LcRgb class (derived from LcColor)
  - RGB color model
  - Input file: (rgb 1.0 0.5 0.5)

- LcRgba (derived from LcRgb)
  - Adds alpha
  - Input file: (rgba 1.0 0.5 0.5 1.0)

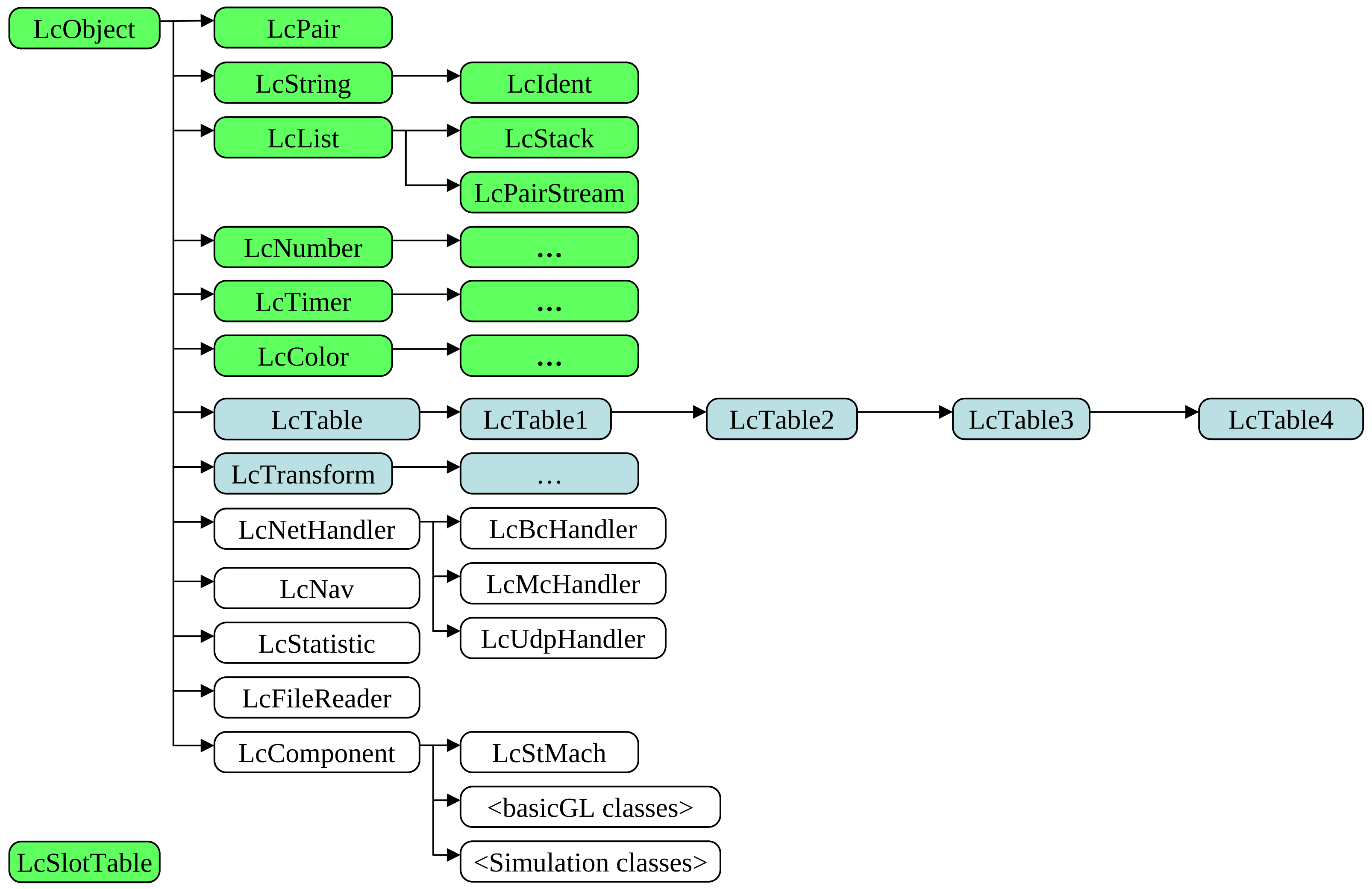
Basic – LcColor (cont)

- LcHsv and LcHsva: Hue, Saturation, Value (HSV) color model (with or without alpha)
  - Easier method to adjust colors than using RGB
    - Example: Color circle GUIs
  - hsv2rgb(osg::Vec4 rgb, osg::Vec4 hsv)
    - Converts HSV to RGB values
  - rgb2hsv(osg::Vec4 hsv, osg::Vec4 rgb)
    - Converts RGB to HSV values
  - Hue is degrees around the color circle
    - Red(0), yellow(60), green(120), cyan(180), blue(240), magenta(300)
  - Saturation: from 0 (white) to 1.0 (fully saturated color)
  - Value: from 0 (black) to 1.0 (full brightness)
  - Input file examples:
    - lightGreen: (hsv hue: 120 saturation: 0.7 value: 1.0)
    - yellow: (hsv 60 1.0 1.0)

Basic – LcColor (cont)

- LcHls: Hue, Lightness, Saturation (HLS) color model
- LcCmy: Cyan, Magenta, Yellow (CMY) color model
- LcCie: Commission Internationale de L'Eclairage (CIE) color model
  - Y primary describes the luminance component
  - x and y describe chrominance components
- LcYiq: YIQ color model
  - Y (luminance) is same as Y primary of CIE standard
  - I and Q are the chrominance components
- LcColorRotary: Determines the object color based on an input value compared against a list of breakpoints paired with a list of colors.
- LcColorRotaryB: Determines the object color based on an input value (0 or 1) when given two selections of type color or ident.

Basic - Class Hierarchy



Basic – LcTable

- Purpose: Abstract class for 1, 2, 3 and 4 dimensional data tables. Provides common methods for loading the data tables, checking table validity and performing linear interpolation.

- Can only be created as a derived class
  - LcTable1 // 1D data table (x)
  - LcTable2 // 2D data table (x, y)
  - LcTable3 // 3D data table (x, y, z)
  - LcTable4 // 4D data table (x, y, z, w)

- Each increase in dimension is derived from the previous class: LcTable3 is derived from LcTable2

- The linear function interpolation (lfi) routines used by the derived table classes are available as static member functions in LcTable
  - LCreal lfi(LCreal x, LCreal* x_data, int nx, LCreal* a_data)

Basic – LcTable (cont)

- LcTable member functions
  - bool isValid()
  - findMinMax(LCreal* minValue, LCreal* maxValue)
  - int tableSize()
  - LCreal* getDataTable()

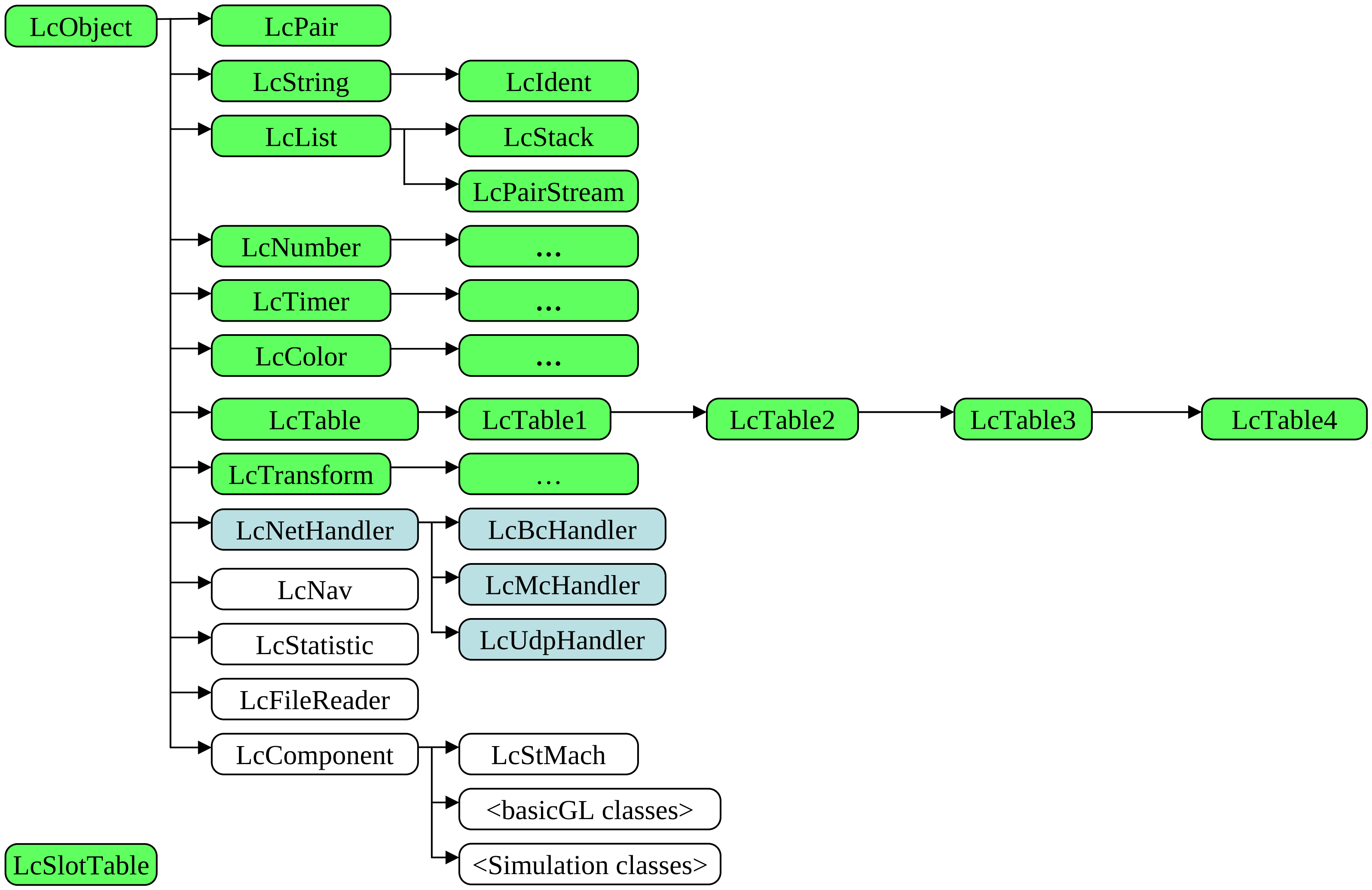
- LcTable1 member functions
  - int getNumXPoints()
  - LCreal* getXData()
  - LCreal getMinX()
  - LCreal getMaxX()
  - LCreal lfi(LCreal ix)

- LcTable2, LcTable3 and LcTable4 member functions
  - Similar to LcTable1 for Y, Z and W

Basic – LcTransform

- LcTransform
  - Base class for various transformations
  - Contains a standard 4x4 transformation matrix
- Derived classes
  - LcTranslation
  - LcRotation
  - LcScale

Basic - Class Hierarchy



Basic – LcNetHandler

- Purpose: General (connectionless) network handler: Can be used for UDP/IP, Multicast and Broadcast.

- Each handler manages a socket, so handlers are needed for input and output.

- Slots:
  - port: Port number
    - send: port we send packets to
    - receive: port we receive packets from
  - sourcePort: Source port number
    - send: port we send packets from
    - receive: identifies applications on send port
  - shared: Set socket's shared (reuse) flag (0: not shared)

Basic – LcNetHandler (cont)

- Derived classes: LcBroadcastHandler, LcMulticastHandler and LcUdpHandler
  - LcBroadcastHandler Slots:

• networkAddr:	Host IP Address
• networkMask:	Host Net Mask

  - LcMulticastHandler Slots:

• multicastGroup:	String containing the multicast IP address
• ttl:	Multicast Time-To-Live (TTL) value
• loopback:	Multicast Loopback flag; default: 0 (off)

  - LcUdpHandler Slots:

• ipAddress:	Destination IP address

- Input File Example:

```
( MyNetworkIO
      netOutput:
          ( BroadcastHandler
               networkAddr: "224.0.0.251"        // Host IP address
               networkMask: "255.255.255.0"   // Host Network Mask
               port: 2010                               // Destination port
               sourcePort: 2                             // Port to send from
               shared: 1                                 // Shared port flag   )   )
```

Basic – LcNetHandler (cont)

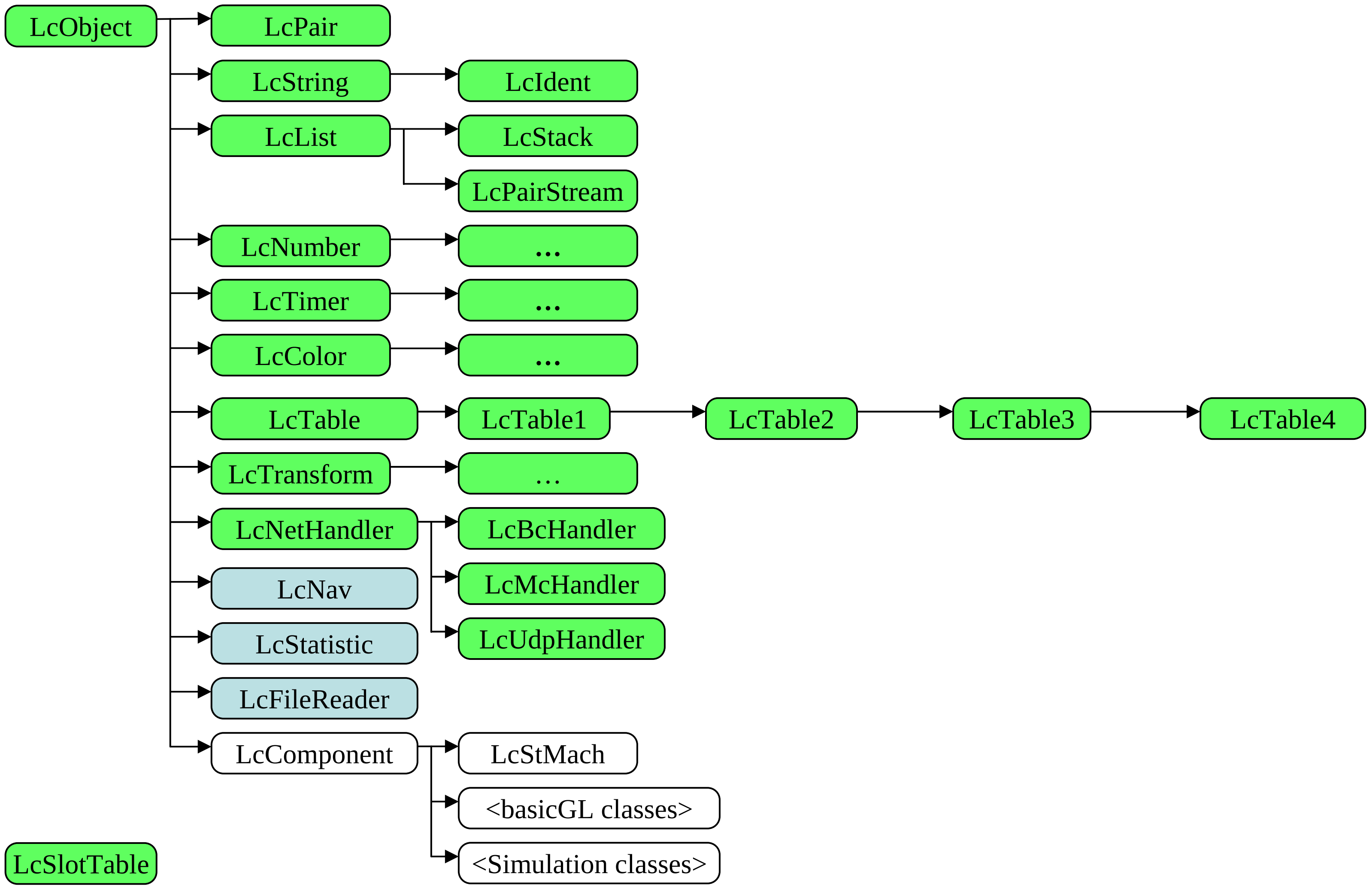
- Network administration and data transfer functions implemented by derived classes.

- Shared port functions
  - bool getSharedFlag()
  - void setSharedFlag(bool b)

- Static functions for network byte swapping/order (all net handlers)
  - toNet(void* hostData, void* netData, int nl, int ns)
  - toHost(void* netData, void* hostData, int nl, int ns)
  - isNetworkByteOrder()
  - isNotNetworkByteOrder()
  - checkByteOrder()

  - toNetOrder(<T>* vout, <T> vin)
  - fromNetOrder(<T>* vout, <T> vin)
    - <T> can be of type: short, unsigned short, long, unsigned long, float or double

Basic - Class Hierarchy



Basic – LcNav

- Purpose: Provide general navigation conversion functions.

- Great-Circle: Lat/Lon to/from Bearing/Distance static functions
  - gbd2ll(…)
  - gll2bd(…)

- Flat-Earth: Lat/Lon/alt to/from position vector [x y z] NED static functions
  - convertPosVec2LL(…)
  - convertLL2PosVec(…)

- Geodetic to/from Geocentric Conversion static functions
  - getGeodCoords(…)
  - getGeocCoords(…)
  - getGeodAngle(..)
  - getGeocAngle(…)
  - getSimPosAccVel(…)
  - getWorldPosAccVel(…)

Basic - Statistic

- Purpose: Provides a general statistics class that acts like a statistical calculator.

- Member functions for computing mean, absolute mean, variance, standard deviation, RMS, maximum and minimum values of all data points added to the statistic.

• sigma(…)	adds data point(s)
• clear()	readies the statistic for new data

Basic - LcFileReader

- Purpose: Manages the reading of the fixed record length files so objects can have direct access to specific records.

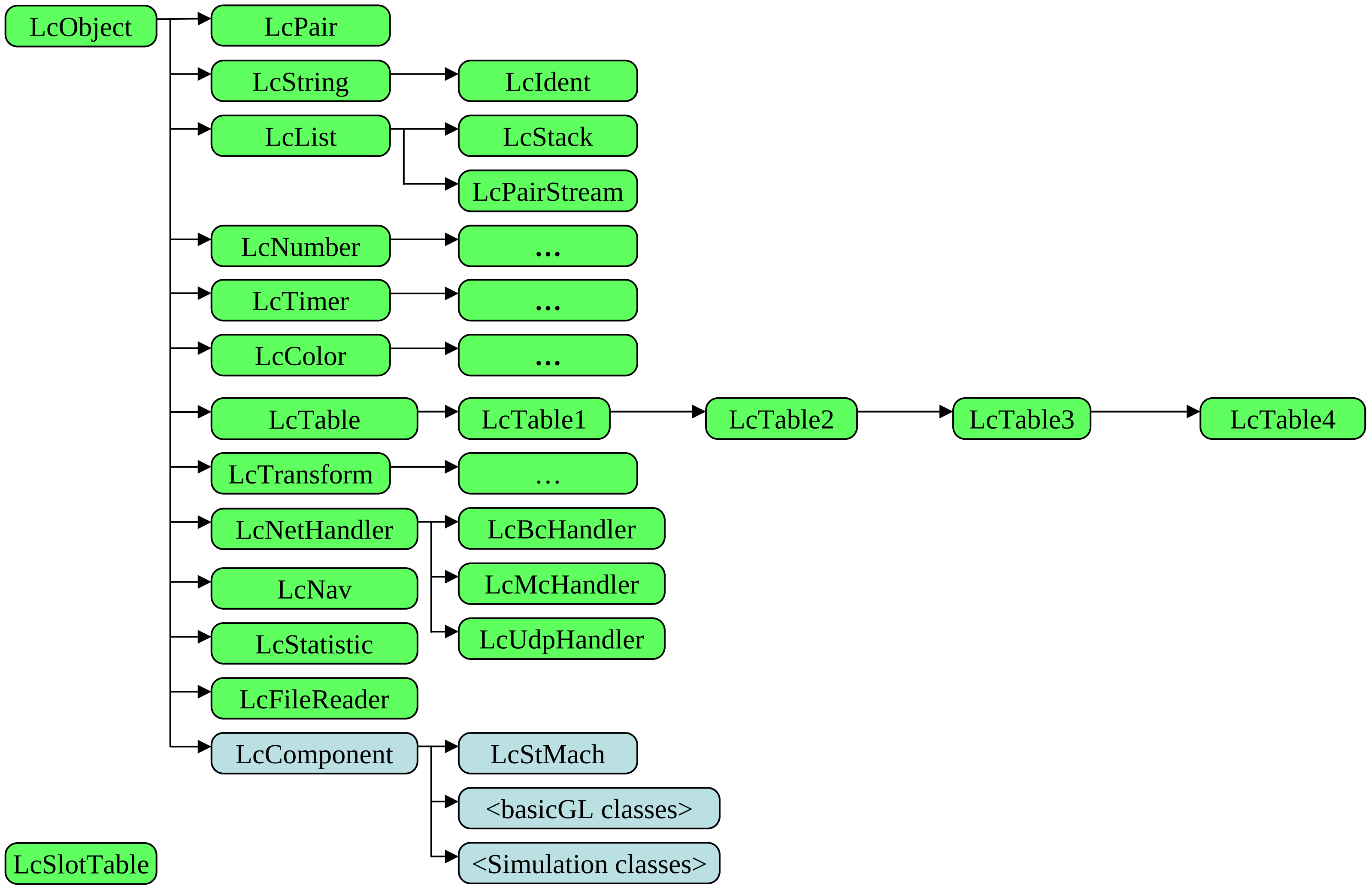
- Slots:
  - 1. pathname (String) Path to the file
  - 2. filename (String) File name (appended to pathname)
  - 3. recordLength (Number) Length (in characters) of the records

- bool open()
  - Opens 'filename' in directory 'pathname‘ with 'recordLength' characters per record.

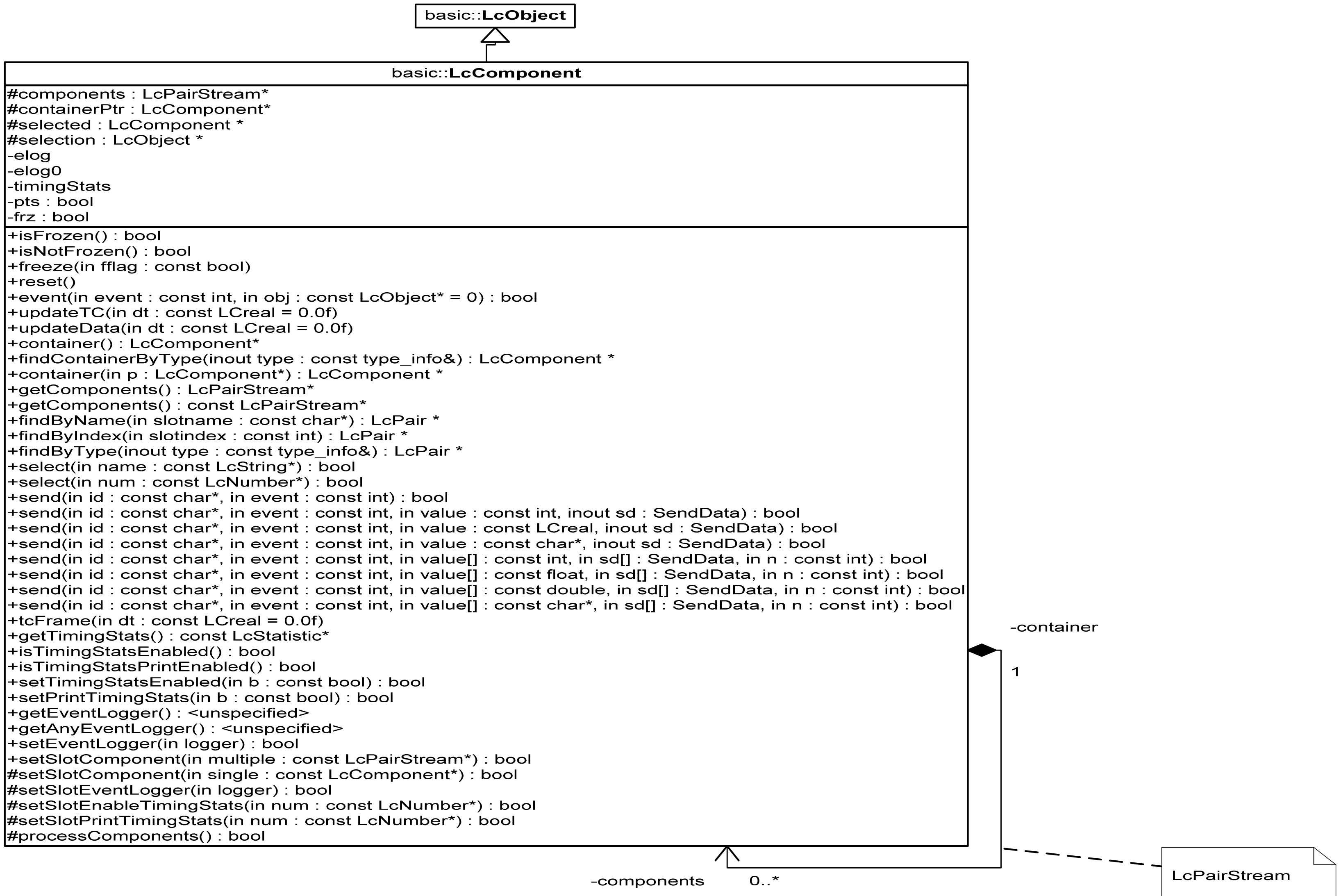
- bool setRecordLength(int n)
  - Sets the current record number to ‘n’
  - Must be called before the first access, if 'recordLength' is not provided to open()

- char* getRecord()
- char* getRecord(int n)
- char* getRecord(int n, int len)
  - Reads current record or ‘len’ characters of record number ‘n’ (if specified)

Basic - Class Hierarchy



LcComponent Static Structure



```
basic::LcComponent
#components : LcPairStream*
#containerPtr : LcComponent*
#selected : LcComponent *
#selection : LcObject *
-elog
-elog0
-timingStats
-pts : bool
-frz : bool
+isFrozen() : bool
+isNotFrozen() : bool
+freeze(in fflag : const bool)
+reset()
+event(in event : const int, in obj : const LcObject* = 0) : bool
+updateTC(in dt : const LCreal = 0.0f)
+updateData(in dt : const LCreal = 0.0f)
+container() : LcComponent*
+findContainerByType(inout type : const type_info&) : LcComponent *
+container(in p : LcComponent*) : LcComponent *
+getComponents() : LcPairStream*
+getComponents() : const LcPairStream*
+findByName(in slotname : const char*) : LcPair *
+findByIndex(in slotindex : const int) : LcPair *
+findByType(inout type : const type_info&) : LcPair *
+select(in name : const LcString*) : bool
+select(in num : const LcNumber*) : bool
+send(in id : const char*, in event : const int) : bool
+send(in id : const char*, in event : const int, in value : const int, inout sd : SendData) : bool
+send(in id : const char*, in event : const int, in value : const LCreal, inout sd : SendData) : bool
+send(in id : const char*, in event : const int, in value : const char*, inout sd : SendData) : bool
+send(in id : const char*, in event : const int, in value[] : const int, in sd[] : SendData, in n : const int) : bool
+send(in id : const char*, in event : const int, in value[] : const float, in sd[] : SendData, in n : const int) : bool
+send(in id : const char*, in event : const int, in value[] : const double, in sd[] : SendData, in n : const int) : bool
+send(in id : const char*, in event : const int, in value[] : const char*, in sd[] : SendData, in n : const int) : bool
+tcFrame(in dt : const LCreal = 0.0f)
+getTimingStats() : const LcStatistic*
+isTimingStatsEnabled() : bool
+isTimingStatsPrintEnabled() : bool
+setTimingStatsEnabled(in b : const bool) : bool
+setPrintTimingStats(in b : const bool) : bool
+getEventLogger() : <unspecified>
+getAnyEventLogger() : <unspecified>
+setEventLogger(in logger) : bool
+setSlotComponent(in multiple : const LcPairStream*) : bool
#setSlotComponent(in single : const LcComponent*) : bool
#setSlotEventLogger(in logger) : bool
#setSlotEnableTimingStats(in num : const LcNumber*) : bool
#setSlotPrintTimingStats(in num : const LcNumber*) : bool
#processComponents() : bool
```

Basic – LcComponent

- Purpose: Components provide a common interface for both time-critical and background tasks, and for passing messages. Any component can be a container for a list of subcomponents and it can be a subcomponent to another container component.

- Subcomponent list
  - Each subcomponent is an LcPair (LcIdent and LcComponent)
  - The subcomponent list is an LcPairStream
  - The following functions are used to locate subcomponents
    - LcPair* findByName(char* slotname)
    - LcPair* findByIndex(int slotindex)
    - LcPair* findByType(type_info& type)

- Container functions
  - LcComponent* container()
    - Returns our container (we are one of its subcomponents)
  - LcComponent* findContainerByType(type_info& type)
    - Finds a container, somewhere up the component tree, of type 'type'.

Basic – LcComponent (cont)

- Event message functions
  - bool event(int token, LcObject* obj = 0)
    - Sends an event token with an optional object
    - Used primarily for message passing (may rename)
    - Implemented with macros

- Event Tokens
  - Standard event tokens are defined in support.h
  - Subsystems can define their own event tokens as needed

- Send functions
  - Alternative method to send events to subcomponents
  - send(char* name, int token, T value, SendData sd)
    - The subcomponent (name) will receive the event token and an object containing the value.
    - Value types, T, can be float, double, char*, and int
    - The SendData structure, sd, keeps a pointer to the subcomponent and remembers the previous value (i.e., event() is only called if the value has changed)
  - Send functions also support an array of values sent to a series of subcomponents

Basic – LcComponent (cont)

- event() can be defined macros

  Example:

  ```
  BEGIN_EVENT_HANDLER(MyClass)
  ON_EVENT_OBJ(RF_EMISSION,onEmissionEvent,Emission)
  ON_EVENT(ON_ENTRY,onEntryEvent)
  ON_EVENT(ON_EXIT,onExitEvent)
  END_EVENT_HANDLER()
  ```

  BEGIN_EVENT_HANDLER() and END_EVENT_HANDLER() define the start and end of the map

- ON_EVENT() maps an event token to a function to handle the event
  - Second argument is the appropriate ‘event’ function to call if the event token matches the first argument

- ON_EVENT_OBJ() maps an event token and object type to a function to handle the event
  - Second argument is the appropriate ‘event’ function to call if the event token matches the first argument and the object type match is the third argument

- On Event functions are implemented for each class
  - Standard set function templates:
    - bool setXXX(T* p)
    - bool setXXX()
  - Each function must error check the argument
  - Return true if the event was processed

Basic – LcComponent (cont)

- Data update functions
  - updateTC(LCreal deltaTime)
    - Performs Time-Critical (real-time) tasks
  - updateData(LCreal deltaTime)
    - Performs background tasks and communicates with graphic displays
  - These function calls are passed on down to all subcomponents
- Reset support
  - Receives the RESET_EVENT message and calls its virtual member function reset()
  - Derived components overload the reset() function, as needed
- LcStMach class
  - Derived from LcComponent
  - General purpose state machine