Robert Felten SOFTWARE, LLC

Introduction to Qt Container Classes

Robert Felten
Independent Software Development Engineer
www.robertfelten.com
robert@robertfelten.com

Contents

Robert Felten
SOFTWARE, LLC

About Robert Felten

- 70s
  - Hughes Aircraft Copany
    - Space and Comm – El Segundo
- 80s
  - Contractor at TRW
    - Space and Comm – Manhattan Beach
- 90s to 2007
  - Raytheon
    - Space and Airborne Systems – El Segundo
- 2007 to 2012
  - Applied Signal Technology
- 2012 to present
  - Independent Contractor – with IDT and PCM-Sierra

Robert Felten
SOFTWARE, LLC

Container Classes

- Mid 90s – learned C++
  - Radar instrumentation
    - Custom library – dynamic arrays, linked lists, strings
      - Didn’t even use templates.
  - Discovered STL
    - Effective C++, Effective STL
    - Began using in all my applications
- 2009 – took over Qt program
  - Horrified to discover they were using Qt custom library
    - QStrings, QLists, QMaps, QHash, etc.
  - First tried to mix existing Qt code with STL containers

Robert Felten
SOFTWARE, LLC

Changed My Mind

- After a while – I discovered
- I LIKED Qt Containers
- Why?

To answer the question

- Why I dumped the STL and Boost libraries
  - And now use Qt container classes exclusively
- Qt Containers have….
  - More intuitive interfaces
  - More powerful built-in functions
  - More efficient implementations
  - More flexible options
  - Great online documentation
  - Sample code and demos
- Note - Not comparing C++ 11 / 14

Container Comparisons

- Comparing STL and Qt containers
  - Most containers in STL have Qt equivalent, and Vice Versa.
  - Most containers have similar constructors, iterators, functions, and algorithms.
  - Qt containers usually have additional constructors, operators, and functions.
    - Also java-style iterators for those so inclined.
  - Qt has added a foreach keyword (implemented in the C++ preprocessor)
    - for efficiently iterating over all members.
  - Some containers are implicitly shared
    - You can pass them by value efficiently.

Color Code for Comparisons

- Comparisons of STL vs Qt
  - Features in common between STL and Qt are in black type.
  - Features unique to Qt are in green type.
  - Features unique to STL are in red type.

Robert Felten SOFTWARE, LLC

Similar Containers

STL Containers	Qt Containers	Internal Structure
vector	QVector	Dynamic Array, adjacent storage
	QList	Dynamic Array
list	QLinkedList	Doubly Linked List
set/multiset	QSet/QMultiset	Sorted list of values
map/multimap	QMap/QMultiMap	Sorted list of key/value pairs
hash (Boost only)	QHash/QMultiHash	Unsorted array of key/value pairs
stack	QStack	Last in First Out dynamic array
queue	QQueue	First in First Out dynamic array
string	QString	Array of characters
	QStringList	QList of QStrings

First choice container

- stl::vector is the usually the most appropriate container
  - Use stl::vector 90% of the time you need dynamic storage.
  - QList should be used even more often (if not quite 100%)

QList<T>

- The workhorse of Qt Container Classes
- Similar to std::vector
  - Fast indexed based access
  - Does not store data in adjacent memory positions
    - If you need adjacent memory, use Qvector.
    - If size of T > size of pointers, stores data as array of pointers
    - Otherwise, stores T itself
  - Fast insertions and removals (see next slide)
  - Not a linked list that guarantees constant time inserts
    - Use QLinkedList

QList Advantages

- Advantages over std::vector
  - More natural syntax for insertions
  - For < 1000 entries, very fast insertions in middle
  - Convenience functions gives more utility.
  - Powerful built-in algorithms
  - Easily convertible to/from other container classes
  - Alternate names and syntax for same functions
    - Gives your code a more natural self-documentation

Robert Felten
SOFTWARE, LLC

Other features of interest

- Memory pre-allocated at both ends.
  - Constant time prepend and append in most cases.
- Constant time access by index
- Direct index just as fast as iterators.
- Includes STL-Style iterators and functions for convenience

Robert Felten SOFTWARE, LLC

Accessing Values in QList

- const T& operator[](int i), same as const T& at(int i)
  - returns value at position i (constant time)
  - assert error if index out of range (in debug mode)
  - if index out of range, STD::vector [ ] returns garbage, at throws exception for if index out of range
  - value(int i)
  - returns value at index i, returns default constructed T if index out of range.
- value(int i, const T defaultValue)
  - returns value at index i, returns defaultValue if index out of range.

Robert Felten SOFTWARE, LLC

Values in Qlist(cont.)

- T & front(), overloaded with const T & front() same as first()
  - returns first entry in the list.
- T & back(), overloaded with const T & back() same as last()
  - returns last entry in the list.
- iterator begin()
  - Returns STL-style iterator pointing to first item in list
- iterator end()
  - Returns STL-style iterator pointing to imaginary item past end of list

Robert Felten SOFTWARE, LLC

Inserting Values in QList

- Inserting values at end
  - QList<T>& operator <<(const QList<T> & other)
    - *a << "Mercury" << "Venus" << "Earth" << "Mars";*
  - *operator +() and +=()*
    - *a += "Mercury" + "Venus" + "Earth" + "Mars";*
  - push_back()
    - *a.push_back("Mercury");*
    - *a.push_back("Venus");*
    - *a.push_back("Earth");*
  - void append(const T &value)
    - *a.append("Mercury");*

Inserting Values (Cont.)

- Inserting in Middle
  - insert(int i, const T &value)
    - inserts value at position i.
  - insert (iterator before, const T &value)
    - inserts value before iterator.

Removing entries from QList

- void pop_front(), removeFirst()
  - removes first entry
- T takeFirst()
  - removes first entry, and also returns it.
- pop_back(), removeLast()
  - removes last entry, does not return it
- T takeLast()
  - removes last entry, and also returns it.

Robert Felten
SOFTWARE, LLC

Removing Entries (cont.)

- removeOne(const T & value)
  - removes first occurrence of value.
- removeAll (const T &value)
  - removes all occurrences of value.
- removeAt (int i)
  - removes element at index i.
- takeAt (int i)
  - removes item at index j, and also returns it.
- removeAll(), same as clear()
  - removes all items from list.

Removing Entries (cont.)

- iterator erase(iterator pos)
  - Removes item at iterator, returns iterator to next entry
- iterator erase(iterator begin, iterator end)
  - Removes items from begin up to but not including end

Robert Felten SOFTWARE, LLC

Swapping Functions in Qlist

- move (int from, int to)
  - moves item from position “from” to position “to”
- replace (int I, const T &value)
  - replaces item at index i with value.
- swap(int i, int j)
  - swaps elements at index positions i and j.

Robert Felten
SOFTWARE, LLC

Additional QList Functions

- Append a QList to the end of a QList
    - *QList<int> a;*
    - *QList<int> b;*
  - void append (const QList<T> &value)
    - *app4.append(b);*
    - *QList<int> app1 = a + b;*

    - *QList<int> app2= a << b;*

    - *QList<int> app3 = a;*
    - *app3+= b;*

    - *QList<int> app4 = a;*

QList Subsets

- Obtain subsets of a QList
  - QList<T> mid (int pos, int length) – returns a list copied from pos, to length or end)
  - Examples:
    - Get first 5 entries:
      - a.mid(0, 5)
    - Get last 5 entries:
      - a.mid(mid.length() – 5)
    - Get 8 entries starting with entry[3] :
      - a.mid(3,8)

QList Built-in Algorithms

- bool contains(const T &value)
  - returns true if QList contains an occurrence of the value
- bool startsWith (const T & value)
  - returns true if QList starts with value.
- bool endsWith(const T &value)
  - returns true if last entry in QList is value
- int indexOf (const T &value, int from = 0)
  - returns index of first occurrence of value.
- int lastIndexof (const T &value, int from) –
  - returns index of last occurrence of value.

QList Conversions

- toSet – converts QList to QSet.
- toStdList – converts QList to std::List
- toVector – converts QList to QVector.
- fromSet – converts QSet to QList
- fromStdList – converts std::list to QList
- fromVector – converts QVector to QList

QList Sizing

- int size(), same as count(), length()
  - returns number of items in the list
- bool isEmpty(), same as empty()
  - returns true if no items in the list
- void reserve(int alloc)
  - Reserves space for alloc elements.

QList Constructors,...

- Constructors
  - QList()
    - creates empty list
  - QList(const QList<T> &other)
    - copy constructor
  - QList(std::Initializer_list<T> args)
    - Only for C++0x compiler
- Equivalence
  - bool operator== (const QList<T> &other
- Assignment
  - T& operator=(const QList<T> &other)

Missing in QList

- Functions from std::vector not in QList
  - Constructor initializing values
  - Assigning n copies of element n
  - Providing your own allocator
  - Getting capacity and max_size
  - accessing values as ordinary array
    - &a[i].
      - If you need this feature, use QVector.

QList Examples

- Some examples of QLists that I used in my SDD manager application
  – Information stored for each device, where number of devices are discovered by program after it starts running.
  – Namespaces (created inside devices) that can be added and deleted by users.
  – Information read from XML files.
  – Data added and deleted by users.
  – Data generated after number of devices discovered.

QMap <T>

- Similar to std::Map
  - Used when associating values with keys.
    - QMap stores (key,value pairs) sorted by key
    - Because STL did not contain a hash table, I tended to always use maps to store (key,value) pairs.
      - Don't use QMap unless you need the pairs stored in key order
    - Use QHash if you don't need keys sorted
- Differences from std::map
  - Remembers multiple values associated with keys
    - Not handled same way as multipmap / QMultiMap
  - Many additional features and functions

Accessing Items in QMap

- T &operator[] (const Key &key)
  - returns value associated with key, as modifiable reference.
  - if map contains no item associated with key, the function inserts a default constructed item into map, and returns a reference to it.
  - if map contains multiple values associated with key, returns reference to most recently inserted value.
- const T operator[] (const Key &key)
  - same as value, except returns a const value instead of reference.

Robert Felten SOFTWARE, LLC

Accessing Items (cont.)

- const T value (const Key & key)
  - returns value associated with key.
  - if no item with key, returns default constructed value.
  - If more than one item with key, returns most recently added value.
- const T value (const Key &key, const T &defaultValue)
  - if no item with key, returns defaultValue
- QList<T> values() const
  - returns a list containing all the values in the map, in ascending order of their keys.
  - if more than one item with same key, all values are included.
- QList<T> values (const Key & key)
  - returns a list containing all values associated with key.

Accessing Items (cont.)

- iterator begin(), overloaded with const_begin()
  - returns iterator or const_iterator to first item in QMap
- iterator end(), overloaded with
  - returns iterator or const_iterator to imaginary item after the last item in the map.
- iterator find(const Key key)
  - returns iterator pointing to item with key key.
  - If multiple items with key, returns iterator pointing to most recently entered value. Other values accessible by incrementing the iterator.
    - QMap<QString, int> map; ... QMap<QString, int>::const_iterator i = map.find("HDR"); while (i != map.end() && i.key() == "HDR") { cout << i.value() << endl; ++i; }

Accessing Items (cont.)

- const Key key(const T &value, const Key &defaultKey) const
  - returns the first key associated with value value, or defaultKey if map does not contain value)
  - linear time, map optimized to for fast lookups by key
- QList<Key> keys() const
  - returns a Qlist containing all the keys in the map
  - duplicate keys occur multiple times in the list.
- QList<Key> uniqueKeys() const
  - returns a list of keys, where each key only occurs once
- iterator lowerBound(const Key &key)
  - returns iterator pointing to first item with key in the map.
  - If key not in map, returns iterator to nearest item with greater key.

Accessing Items (cont.)

- iterator upperBound (const Key & key)
  - returns iterator pointing to item that immediately follows the last item with key.
  - if map does not contain key, returns iterator to nearest item with a greater key.

Inserting Items in QMap

- iterator insert(const Key &key, const T & value)
  - inserts a new item with key key, and value of value.
  - If there is already an item with key, value is replaced.
- interator insertMulti(const Key &key, const T &value)
  - same as insert, except if already an item with key, adds a new value associated with key.

Removing Items from QMap

- clear()
  - removes all items in map
- iterator erase (iterator pos)
  - removes the (key,value) pair pointed to by pos, returns iterator to next item in map.
- int remove(const Key & key)
  - Removes all items that have key in map. Returns number of items removed
- T take (const Key& key)
  - Removes the item with key, and returns the value.
  - If multiple values, only most recent is removed and returned.

Robert Felten
SOFTWARE, LLC

QMap Extra Functions

- void swap(QMap<Key, T> &other
  - swaps map with other. Very fast / guaranteed not to fail.
- QMap<Key, T> &unite() (const QMap<Key, T> &other
  - Inserts all the items in other map into this map.

Robert Felten
SOFTWARE, LLC

QMap Constructors

- QMap()
  - default constructor – empty map
- QMap(const QMap<Key, T> & other)
  - copy constructor
- QMap(const std::map<Key, T> &other)
  - converts from STL Map

Robert Felten
SOFTWARE, LLC

Qmap Sizing, operators

- int count(), same as size()
  - returns number of items (key/value pairs) in the map.
- isEmpty(), same as empty()
  - returns true if map contains no items.
- bool operator== (const QMap<Key, T) &other)
  - returns true if other is equal to this map, i.e. contain the same (key,value) pairs.
- bool operator!= (const QMap<Key, T>& other)
  - returns true if other is not equal to this map
- QMap<key, T> & operator= (const QMap<Key, T> &other)
  - assigns QMap other to this one.

QMap Built-in Algorithms

- bool contains (const Key &key)
  - returns true if map contains an item with key key
- int count (const Key &key)
  - returns number of items associated with key

QMap Conversions

- std::map<key, T> toStdMap() const
  - converts QMap to std::map.

Missing in QMap

- map (op) – constructor using op as sorting criteria
- max_size()
- operators <, <=, >, >=
- equal_range algorithm
- rbegin, rend (reverse iterators)
- insert (pos, elem) – pos is a hint where to start search
- erase (beg, end) – erase items from beg to end

QMap Examples

- map that associates strings with enumerated values (rather than enumerated values with strings)
- map that associates revision ID with hardware devices

QHash <T>

- QHash stores keys in arbitrary order
  - Use instead of QMap when order of entries does not matter
  - QHash provides faster lookups than QMap, because QMap stores (key,value) pairs in sorted order by key
  - Automatically expands or shrinks table to provide fast lookups without wasting too much memory

Implicitly Shared Container

- QHash is implicitly shared
  - You can copy a QHash table, or return by value from a function – very fast. No actual copy is done.
  - Only when a shared instance is modified will it be copied – in linear time.

Diffs between QMap and QHash

- QList<T> values() const
  - returns a list containing all the values in the hash, ***in arbitrary order.***
- QList<Key> keys(const T & value)
  - returns list containing all keys associated with value *value*
  - ***in arbitrary order***
  - Slow (linear time)
- iterator erase (iterator pos)
  - removes the (key,value) pair pointed to by pos, returns iterator to next item in hash.
    - ***Can be safely called while iterating***

QHash Fine Tuning

- These functions control the QHash internal table
  - Use rarely, if ever.
    - QHash automatically shrinks or grows for good performance
  - int capacity()
    - returns number of buckets in internal hash.
  - void reserve(int size)
    - ensure that QHash internal hash table contains at least size buckets.
    - Used to avoid repeated allocation for large hash tables.
  - squeeze()
    - Reduces size of internal hash table to save memory

QHash Operators

- bool operator== (const QHash<Key, T) &other)
  - returns true if other is equal to this hash, i.e. contain the same (key,value) pairs.
- bool operator!= (const QHash<Key, T>& other)
  - returns true if other is not equal to this hash
- QHash<key, T> & operator= (const QHash<Key, T> &other)
  - assigns QHash other to this one.

Controlling Hash Keys

- uint qHash(type key)
  - returns the hash value for key
  - Note – this function is overloaded for all the different types, i.e.
    - char, uchar, signed char, ushort, short, uint, int, ulong, etc.)

QStack

- Derived from a QVector
  - Implements Last In / First Out (LIFO)
  - Has all the capabilities and functions of a QVector, plus the following:
    - ***T pop()***
      - Removes the top item from the stack and returns it.
    - ***void push(const T & t)***
      - Adds element t to the top of the stack.
      - This is the same as *QVector::append().*
    - ***T &top() / const T &top() const***
      - Returns a reference to the stack's top item.
      - This is the same as QVector::last().

QQueue

- Derived from a QList.
  - Implements First In / First Out
    - QList already does that anyway
    - New functions added only for convenience
  - Contains all the functions and features of a QList, plus the following:
    - ***T dequeue()***
      - *r*emoves the head item in the queue and returns it.
      - This is the same as *QList::takeFirst()*
    - ***void enqueue(const T &t)***
      - Adds value t to the tail, same as *QList::append()*
    - ***T &head(), const T & head() const***
      - same as *QList::first()*

QString

- QStrings are the “elephant in the room”
  - We haven’t talked about it yet
- It’s simply the most amazing, powerful, versatile, usable string class I’ve ever seen, or hope to see
- Related Containers
  - QString – basic string
  - QStringList – essentially QList<QString>
  - QByteArray – array of bytes that can be null terminated char* or contain 0s.
  - QChar – 2-byte character

QString

- stl::string contains an array of 1-byte char.
- QString contains an array of 2-byte QChars
  - each representing one 4.0 Unicode character.
    - Unicode supports international standard characters
  - If you need an array of raw bytes, you can use QByteArray
  - functions are available to convert QString to ASCII, Latin1, Utf8 or Local8Bit (which converts to the system's local environment)

QString Construction

- Construct from QChar*, Qchar, char*, char, QByteArray, and others
- append from various sources
- arg() constructs strings from other types
  - Example
  - *arg(int a, int fieldWidth, int base, const QChar fillChar)*
    - *int value = 100;*
    - *QString abc = QString("This is value as integer %1, as hex 0x%2, as octal %3h, and as binary%4b")*
      .arg(value)
      .arg(value, 0, 16)
      .arg(value, 0, 8)
      .arg(value, 0, 2);
      - abc = "This is value as integer 100, as hex 0x64, as octal 144h, and as binary 1100100b"

More constructors

- setNum(type, base)
  - QString a;
  - int value = 5;
  - a.setNum(value, 16);
- QString number(type n, char format, int precision);
  - overloaded for all number types, int, double, etc.
  - format e, E, f, g, G

Substrings

- Substrings
  - chop(int n)
    - returns n chars from end of string
  - mid (int pos, int n)
    - returns n characters starting at pos (n = -1, default, returns to end of string)
  - simplified()
    - removes beginning and ending whitespace, and internal multiple whitespace characters

Really cool substrings

- QStringList split (QString sep, behavoir)
  - Splits the string into a QStringList of substrings whenever sep appears.
    - behavior indicates whether case of sep should be matched
    - note – deletes the separator in the QStringList
  - Also overloaded to accept a regular expression instead of a QString
- section(Qstring sep, int start, int end, flags)
    - extracts sections of a string, separated by sep
- QString QStringList::join(QString sep)
  - Combines strings in a QStringList to a single QString (inserting sep between each)

Conversions

- Conversions from strings
  - toInt(), toDouble(), toLongLong(), etc.
  - data() – returns QChar*

Functions

- compare()
- contains(Qstring, char, etc.)
- endsWith()
- indexOf()
- lastIndexOf()
- length()

Manipulations

- fill()
- leftJustified()
- insert()
- append()
- prepend()
- replace()
- rightJustified()
- clear()

Robert Felten
SOFTWARE, LLC

QString Operators

- All the usual suspects
  - +, +=, ==, =, <, <=, >, >=, <<
- All the STL-style iterators are there

Qt Resources

- Just type Qt into Google.
- qt-project.org is the main home page
  - Provides Downloads of the SDK
    - Open source or license version includes technical support
    - The download contains compilable and runnable sample demos for almost every aspect of Qt
    - You can use a demo as the basis for writing your own software
  - Tutorials
  - Forum and Wiki
  - Bug Reports

Questions

- Questions?