Coding Conventions

Formatting

make format-code

I'm not even slightly happy about the way this looks but I've found no better alternative. At least its automated and consistent. It can be configured to use astyle or clang-format, but I've found clang-format slightly less buggy.

STL naming versus "Studly Caps" or "CamelCase"

I personally prefer the style "CamelCase" – probably because I first did object oriented programing in Object Pascal/MacApp – a few years back. Maybe there is another reason. But now it's a quite convenient – providing a subtle but readable visual distinction.

All (or nearly all) Stroika classes, and methods use essentially the same 'Studly Caps' naming styled from MacApp, with a few minor versions:

However, STL / stdc++ - has its own naming convention (basically all lower case, and _), plus its own words it uses by analogy / convention throughout (e.g. begin, end, empty).

Stroika methods will start with an upper case letter, EXCEPT in the case where they method mimics for follows an existing STL pattern. If you see lower case, assume the function follows STL semantics. If you see CamelCase, you can assume it follows Stroika semantics.

For example:

String:: SubString () follows Stroika semantics (asserting if values out of range).

String::substr() follows the semantics of STL's basic_string<>::substr().

Note – this 'convention' doesn't replace documentation (the behavior of each method is documented). It just provides the user/reader a quick subtle convenient visual cue which semantics to expect without reading the docs.

Examples of common STL methods which appear in Stroika code (with STL semantics):

- clear()
- empty()
- push_back
- erase
- c_str()
- length()
- size()
- compare()
- begin() // sort of – usually using Stroika iterators
- end() // ditto

Name Prefixes and Suffixes

Prefixes

- the 'f' prefix for data members
- 'k' prefix for constants
- We use the 'e' prefix for enumerators
- 't' prefix for thread_local variables
- 's' prefix for static varaibles.
- '_' prefix for PROTECTED instance variables or functions

Suffix

- '_' suffix for PRIVATE instance variables or functions

Begin/End versus start/length

STL is reasonably consistent, with most APIs using T* start, T* end, but some APIs use length instead of end. The Stroika convention is to always use T* start, T* end.

Rationale

One, this gives more consistent expectations. That's especially important for APIs that use offsets (like String) – so that it's obvious the meaning of integer parameters.

And it avoids problems with overflow. For example, if you had an API like:

```
basic_string substr(
   size_type _Off = 0,
   size_type _Count = npos
) const
```

To map this to an internal representation you have todo:

```
char* s = m_bufPtr + _Off;
char* e = m_bufPtr + _Off + _Count;
```

but if count was numeric_limits<size_t>::max(), then the e pointer computation would overflow. There are ways around this, but mixing the two styles creates a number of problems - but for implementations – and for use.

New static methods and Factories

In Stroika, a New () is static method, which allocates an instance of some class, but returns some kind of shared_ptr/smart pointer to the type – not a bare C++ pointer.

Stroika doesn't make much use of the factory pattern, but occasionally – it is useful. If the type provided by the factory is exactly the type of a given class, then we generally use

```
struct T_Factory {
    static T New();
};
```

That technique is used to control the default kind of containers (backend algorithm) that is used.

Or for Stream classes, the 'stream quasi namespace' contains a New method to construct the actual stream, and the definition of the Ptr type – smart pointer – used to access the stream.

Compare () and operator<, operator>, etc…

For types Stroika defines, it generally uses the convention of providing a compare function:

```
int T::Compare (T rhs);    // sometimes with additional optional
                           // arguments for how to compare
```

and provides

bool operator<, operator<=, operator>, operator>=, operator==, operator!= which inline trivially maps to this.

Stroika code which COUNTS on comparison doesn't directly call Compare(), but instead uses 'a < b', etc. This applies to things like Stroika containers. The reason for this later choice include:

- Working with builtin types (e.g. in)
- Working with STL types, and 3rd-party libraries
- Probably more likely to seamlessly fit with user code

Note that we choose to use non-member operator overloads for these comparison functions because putting them in the namespace where the class is defined provides the same convenience of use (name lookup) as member functions, but allows for cases like C < O where C is some time convertible to O, and O is the class we are adding operator< support for.

So for example:

```
if (L"aa" < String (L"ss")) {
}
```

Works as expected, so long as either the left or right side is a String class, and the other side is convertible to a String.

Using T= versus typedef

C++11 now supports a new typedef syntax – using T=…. This is nearly the same as typedef in terms of semantics.

Stroika code will generally use the using T = syntax in preference to typedef for two reasons:

- The using = syntax is slightly more powerful, in that it supports defining derivative template typdefs.
- And more importantly, I believe it makes code more readable, because the type of INTEREST is the one being defined = which appears first. What it maps to is often more complicated (why we define the typedef) – and one can often ignore that detail (or skim it).

Procedure name suffixes for string return type

The Windows SDK uses the convention of appending a W to the end of a function name that uses wide characters, and an A to the name that uses the current operating system locale for code page.

In C++ (and Stroika) – this convention is also generally unneeded, because of the availability of overloading.

Stroika generally avoids this issue by returning String classes nearly everywhere – which are Unicode based. But as the Stroika String classes uses the rest of the Stroika infrastructure – including thread interruption, it's sometimes inconvenient for some low level coding to use those String classes.

But you cannot overload on return types.

For this reason, a handful of Stroika APIs follow the convention of a suffix of:

- _U for wstring return
- _N for string return, being interpreted as the 'narrow SDK code page'
- _A for string returns which are guaranteed to be ASCII

kThe for some final singleton objects

Some objects which are only usable after the start of main (and until end of main), may be slightly more convenient and performant to use pre-existing ones. For example, `EOFException::kThe`, `InterruptException::kThe`, etc.