```
        {
        private:
            std::uint64_t d_securityHash;
            std::uint32_t d_key;
                // sizeof(Key) is 16 bytes.

        public:
            std::uint32_t key() const;  // stable interface function

            // ...
        };
    }
}
```

Attempting to link together older binary artifacts built against version 1 with binary artifacts built against version 2 will result in a link-time error rather than allowing an ill-formed program to be created. Note, however, that this approach works only if functionality essential to typical use is defined out of line in a `.cpp` file. For example, it would add absolutely no value for libraries that are shipped entirely as header files, since the versioning offered here occurs strictly at the binary level (i.e., between object files) during the link stage.

Build modes and ABI link safety

In certain scenarios, a class might have two different memory layouts depending on compilation flags. For instance, consider a low-level `ManualBuffer` class template in which an additional data member is added for debugging purposes:

```
template <typename T>
struct ManualBuffer
{
private:
    alignas(T) char d_data[sizeof(T)];  // aligned and big enough to hold a T

#ifndef NDEBUG
    bool d_engaged;  // tracks whether buffer is full (debug builds only)
#endif

public:
    void construct(const T& obj);
        // Emplace obj. (Engage the buffer.) The behavior is undefined unless
        // the buffer was not previously engaged.

    void destroy();
        // Destroy the current obj. (Disengage the buffer.) The behavior is
        // undefined unless the buffer was previously engaged.

    // ...
};
```