

C++ Tips and Techniques with BREW®

Paul L. Anderson, Author and Lecturer
Anderson Software Group, Inc.



Why C++?

- **Benefits**
 - Data abstraction and encapsulation
 - C++ has better type checking than C
 - Good for OOP and reusability
- **Using C++ with BREW**
 - Classes, interfaces
 - Constructors, destructors
 - Class member functions
 - Inheritance
 - Virtual functions
 - Templates

C++ Design Approach

- **Similar to BREW Application Wizard**
- **“Boilerplate” C++ Files**
 - Header File
`MyApp.h`
 - Source File
`MyApp.cpp`
- **User C++ Files**
 - Header File(s)
`Object.h`
`MyObject.h`
 - Source File(s)
`MyObject.cpp`

MyApp.h

```
#ifndef _MYAPP_H__
#define _MYAPP_H__

// MyApp.h - header file for C++ App
#include "AEEModGen.h" // Module interface definitions
#include "AEEAppGen.h" // Applet interface definitions
#include "AEEShell.h" // Shell interface definitions
#include "AEEStdLib.h" // Helper functions

// include object header files here...
#include "MyObject.h"

// overload new and delete operators
inline void *operator new(size_t size)
    { return MALLOC(size); }
inline void *operator new[](size_t size)
    { return MALLOC(size); }
inline void operator delete(void *ptr) { FREE(ptr); }
inline void operator delete[](void *ptr) { FREE(ptr); }
```

MyApp.h (continued)

```
class MyApp : public AEEApplet {
private:
    AEEDeviceInfo    DeviceInfo;        // device info
    IDisplay         *pIDisplay;
    IShell          *pIShell;
    // add any Interface or Object pointers here...
    MyObject        *pObject;
protected:
    // event and message handlers from a C applet
    bool            OnAppInitData();
    void            OnAppFreeData();
    bool            OnEvent(AEEEvent eCode, uint16 wParam,
                           uint32 dwParam);
};
```

MyApp.h (continued)

```
public:
// class static member function definitions
static bool InitAppData(MyApp *pMyApp) {
    return pMyApp->OnAppInitData();
}

static void FreeAppData(MyApp *pMyApp) {
    pMyApp->OnAppFreeData();
}

static bool HandleEvent(MyApp *pMyApp, AEEEvent eCode,
                        uint16 wParam, uint32 dwParam)
    { return pMyApp->OnEvent(eCode, wParam, dwParam); }
};
#endif
```

MyApp.cpp

```
#include "MyApp.h"
#include "MyApp.bid"
extern "C"
int AEEClsCreateInstance(AEECLSID ClsId, IShell *pIShell,
                        IModule *po, void **ppObj) {
    *ppObj = NULL;
    if (ClsId == AEECLSID_MYAPP) { // Create App
        if (AEEApplet_New(sizeof(MyApp), ClsId, pIShell, po,
                        (IApplet**)ppObj,
                        // pass pointers to static member functions
                        (AEEHANDLER)MyApp::HandleEvent,
                        (PFNFREEAPPDATA)MyApp::FreeAppData)) {
            // Initialize applet data
            if (MyApp::InitAppData((MyApp*)*ppObj)) {
                return AEE_SUCCESS;
            } else {
                IAPPLET_Release((IApplet*)*ppObj);
                return EFAILED;
            }
        }
    } // end AEEApplet_New
}
return EFAILED;
}
```

MyApp.cpp (continued)

```
// class member function definitions
bool MyApp::OnAppInitData() {
    // initialize Display and Shell variables
    pIDisplay = m_pIDisplay;
    pIShell = m_pIShell;

    // Get the device information for this handset
    DeviceInfo.wStructSize = sizeof(DeviceInfo);
    ISHELL_GetDeviceInfo(pIShell, &DeviceInfo);

    // Insert code here for creating objects here...
    pObject = new MyObject(pIShell);
    // check for errors, return false if found
    return true;
}

void MyApp::OnAppFreeData() {
    // Insert code for deleting objects here...
    delete pObject;
}
```

MyApp.cpp (continued)

```
// class member function definitions
bool MyApp::OnEvent(AEEEvent eCode, uint16 wParam,
                   uint32 dwParam) {
    switch (eCode) {
        // App is told it is starting up
        case EVT_APP_START:
            // Add your code here...
            return true;

            // App is told it is exiting
        case EVT_APP_STOP:
            // Add your code here...
            return true;

            // other events here...
    }
    return false;
}
```

Object.h

```
#ifndef _OBJECT_H__
#define _OBJECT_H__
// Object.h - Object abstract base class (superclass)

class Object {
public:
    virtual ~Object() { }          // virtual destructor
    virtual const char *myType() const = 0;    // type id
};
#endif
```

MyObject.h

```
#ifndef _MYOBJECT_H__
#define _MYOBJECT_H__
// MyObject.h - header file for object
#include "AEEStdLib.h" // Helper Functions
// include files here for interfaces...
#include "Object.h" // Object superclass

class MyObject : public Object {
private:
    // instance variables here
    IShell *pIShell;
    . . .
public:
    MyObject(IShell *ps) { /* constructor code here */ }
    ~MyObject() { /* destructor code here */ }
    // member functions here...
    virtual const char *myType() const { return "MyObject"; }
};
#endif
```

Double Object

- **Objectives**

- Encapsulate primitive `double` type
- Implement floating point operations
- Make seamless, implicit conversions

- **Examples**

```
Double m = 5.67;   Double n = 3.45;
Double p = m + n;   // Floating add
Double q = m - n;   // Floating subtract
Double r = m * n;   // Floating multiply
Double s = m / n;   // Floating divide
double t = m + 3.5; // implicit conversions
```

Double.h

```
#ifndef __DOUBLE_H__
#define __DOUBLE_H__
// Double.h - header file for object
#include "AEEStdLib.h"          // Helper Function definitions
#include "Object.h"            // Object superclass

class Double : public Object {
private:
    double value;
public:
    Double(double d) { value = d; }
    operator double() const { return value; }
    virtual const char *myType() const { return "Double"; }
};
```

Double.h (continued)

```
inline double operator+(const Double & d1, const Double & d2) {
    return FADD(d1, d2);
}
inline double operator-(const Double & d1, const Double & d2) {
    return FSUB(d1, d2);
}
inline double operator*(const Double & d1, const Double & d2) {
    return FMUL(d1, d2);
}
inline double operator/(const Double & d1, const Double & d2) {
    return FDIV(d1, d2);
}
// conversions
inline double operator+(const Double & d1, double d2)
    { return FADD(d1, d2); }
inline double operator+(double d1, const Double & d2)
    { return FADD(d1, d2); }
. . .
```

SoundPlayer Object

- **Objectives**
 - Encapsulate BREW `IMedia` API
 - Integrate into C++ Boilerplate Files
 - Provide OO interface, make reusable
- **Example – Play MP3**
 - Boilerplate Files
 - `PlayMP3.h`
 - `PlayMP3.cpp`
 - Object Files
 - `Object.h`
 - `SoundPlayer.h`

SoundPlayer Object

- **Example**

```
#include "SoundPlayer.h"    // SoundPlayer Object

// create SoundPlayer object
SoundPlayer *pSoundPlayer =
    new SoundPlayer("music.mp3", pIShell);

pSoundPlayer->start();    // start SoundPlayer
pSoundPlayer->stop();    // stop SoundPlayer

// release SoundPlayer object
delete pSoundPlayer;
```

SoundPlayer.h

```
#ifndef __SOUNDPLAYER_H__
#define __SOUNDPLAYER_H__
// SoundPlayer.h - header file for SoundPlayer object
#include "AEEShell.h"           // Shell interface definitions
#include "AEEMediaUtil.h"      // Media Interface definitions
#include "AEEStdLib.h"         // Helper functions
#include "Object.h"           // Object superclass

class SoundPlayer : public Object {
private:
    const char    *pTune;           // MP3 filename
    IShell        *pIShell;        // Shell interface
    IMediaUtil    *pIMediaUtil;    // IMediaUtil interface
    IMedia        *pIMedia;        // Media interface
    bool          constructed;     // object constructed
};
```

SoundPlayer.h (continued)

public:

```
SoundPlayer(const char *pt, IShell *ps) {
    pTune = pt;  pIShell = ps;
    constructed = false;
    pIMediaUtil = NULL; pIMedia = NULL;
    if (ISHELL_CreateInstance(pIShell, AEECLSID_MEDIAUTIL,
        (void **)&pIMediaUtil) != SUCCESS)
        return;
    AEEMediaData MediaData;
    MediaData.clsData = MMD_FILE_NAME;  // set media source
    MediaData.pData = (void *)pTune;
    if (IMEDIAUTIL_CreateMedia(pIMediaUtil,
        &MediaData, &pIMedia) != SUCCESS)
        return;
    constructed = true;
}
```

SoundPlayer.h (continued)

```
bool objectBuilt() const { return constructed; }

virtual ~SoundPlayer() {
    if (pIMedia) { stop(); IMEDIA_Release(pIMedia); }
    if (pIMediaUtil)
        IMEDIAUTIL_Release(pIMediaUtil);
}

const char *getTuneName() const { return pTune; }
int start() const { return IMEDIA_Play(pIMedia); }
int stop() const { return IMEDIA_Stop(pIMedia); }
int suspend() const { return IMEDIA_Pause(pIMedia); }
int resume() const { return IMEDIA_Resume(pIMedia); }
virtual const char *myType() const { return "SoundPlayer"; }
};

#endif
```

PlayMP3.h

```
#ifndef _PLAYMP3_H__
#define _PLAYMP3_H__
// PlayMP3.h - header file for C++ App
#include "AEEModGen.h" // Module interface definitions
#include "AEEAppGen.h" // Applet interface definitions
#include "AEEShell.h" // Shell interface definitions
#include "AEEStdLib.h" // Helper functions
#include "SoundPlayer.h" // SoundPlayer Object
. . .
// MyApp class replaces myapp_t struct
class MyApp : public AEEApplet {
private:
    AEEDeviceInfo DeviceInfo; // hardware device info
    IDisplay *pIDisplay;
    IShell *pIShell;
    SoundPlayer *pSoundPlayer; // SoundPlayer object
. . .
};
#endif
```

PlayMP3.cpp

```
// PlayMP3.cpp - play an MP3 file
#include "PlayMP3.h"
#include "PlayMP3.bid"
extern "C"
int AEEClsCreateInstance(AEECLSID ClsId, IShell *pIShell,
                        IModule *po, void **ppObj) {
    *ppObj = NULL;
    if (ClsId == AEECLSID_PLAYMP3) { // Create App
        if (AEEApplet_New(sizeof(MyApp), ClsId, pIShell, po,
            . . . .
            // pass pointers to static member functions
            (AEEHANDLER)MyApp::HandleEvent,
            (PFNFREEAPPDATA)MyApp::FreeAppData)) {
            // Initialize applet data
            if (MyApp::InitAppData((MyApp*) *ppObj)) {
                . . . .
            }
        } // end AEEApplet_New
    }
    return EFAILED;
}
```

PlayMP3.cpp (continued)

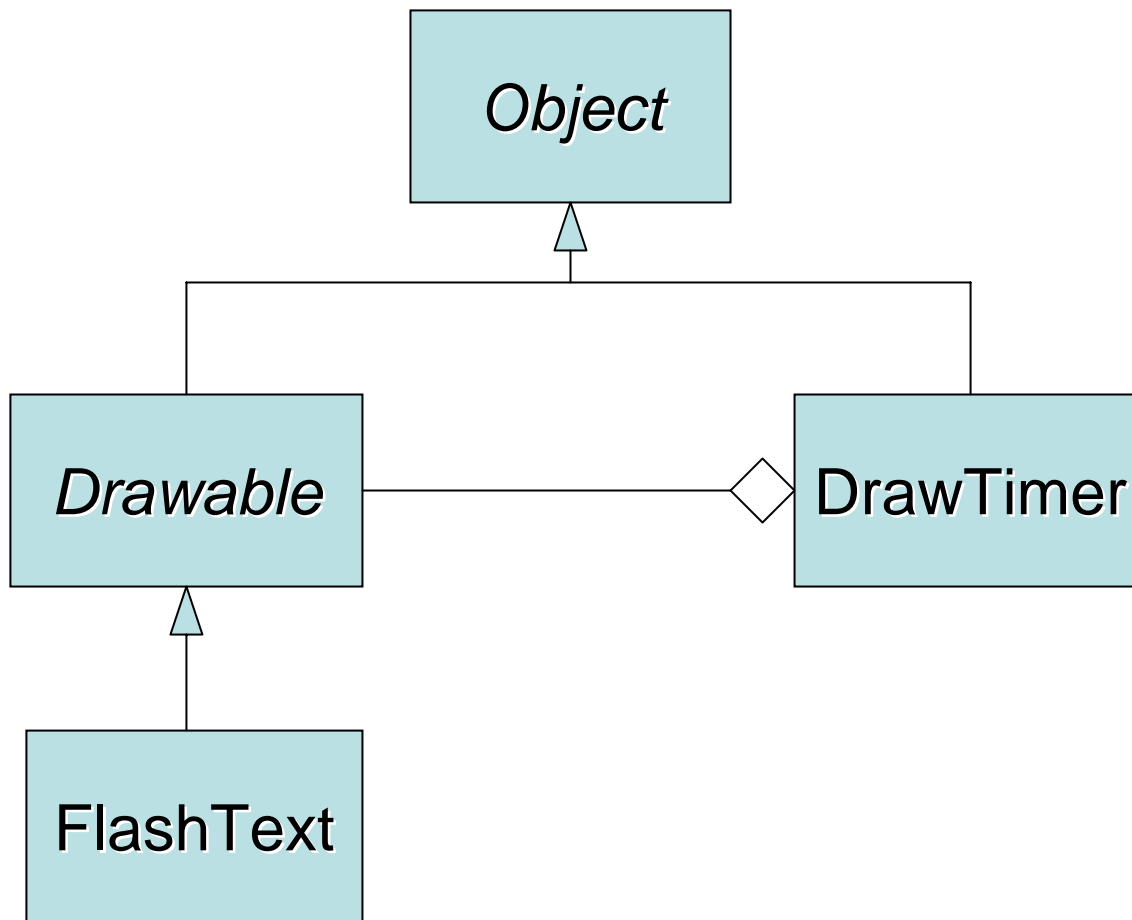
```
bool MyApp::OnAppInitData() {
    . . .
    // create SoundPlayer object to play MP3 file
    pSoundPlayer = new SoundPlayer("music.mp3", pIShell);
    if (pSoundPlayer == NULL || !pSoundPlayer->objectBuilt()) {
        DBGPRINTF("Unable to build SoundPlayer Object");
        return false;
    }
    return true;
}

void MyApp::OnAppFreeData() {
    delete pSoundPlayer;    // delete SoundPlayer object
}
```

PlayMP3.cpp (continued)

```
bool MyApp::OnEvent(AEEEvent eCode, uint16 wParam,
                    uint32 dwParam) {
    switch (eCode) {
        case EVT_APP_START:
            pSoundPlayer->start();
            return true;
        case EVT_APP_STOP:
            pSoundPlayer->stop();
            return true;
        case EVT_APP_SUSPEND:
            pSoundPlayer->suspend();
            return true;
        case EVT_APP_RESUME:
            pSoundPlayer->resume();
            return true;
        . . .
    }
    return false;
}
```

DisplayTimer Example



DrawTimer Object

- **Objectives**
 - Use **Drawable** interface
 - Encapsulate BREW **SetTimer/CancelTimer** APIs
 - Repaint screen periodically, make reusable
- **Example – DisplayTimer**
 - Object Files
 - Object.h**
 - Drawable.h**
 - DrawTimer.h**
 - FlashText.h**
 - SoundPlayer.h**

Drawable.h

```
#ifndef _DRAWABLE_H__
#define _DRAWABLE_H__
// Drawable.h - Drawable interface
#include "Object.h"

class Drawable : public virtual Object {
public:
    virtual void paint() = 0;
};
#endif
```

DrawTimer.h

```
#ifndef __DRAWTIMER_H__
#define __DRAWTIMER_H__
// DrawTimer.h - DrawTimer object
#include "AEEStdLib.h"           // Helper Function definitions
#include "Object.h"             // Object superclass
#include "Drawable.h"          // Drawable interface

class DrawTimer : public Object {
private:
    int time;                   // time between repaints
    Drawable *pDrawable;        // object to repaint
    IShell *pIShell;           // IShell interface
    bool constructed;           // object constructed
protected:
    void OnTimerCB();
    static void TimerCB(DrawTimer *pMe) { // timer callback
        pMe->OnTimerCB();
    }
}
```

DrawTimer.h (continued)

public:

```
    DrawTimer(int tm, Drawable *pd, IShell *ps) {
        time = tm;        pDrawable = pd;
        pIShell = ps;    constructed = true;
    }
    void setTime(int newTime) { time = newTime; }
    int getTime() const { return time; }
    int start() { return ISHELL_SetTimer(pIShell, time,
        (PFNNOTIFY)DrawTimer::TimerCB, this); }
    int stop() { return ISHELL_CancelTimer(pIShell,
        (PFNNOTIFY)DrawTimer::TimerCB, this); }
    bool objectBuilt() const { return constructed; }
    virtual const char *myType() const { return "DrawTimer"; }
};

inline void DrawTimer::OnTimerCB() {
    pDrawable->paint();    // virtual call to redraw
    start();              // restart timer
}
#endif
```

FlashText.h

```
#ifndef _FLASHTEXT_H__
#define _FLASHTEXT_H__
// FlashText.h - FlashText object
#include "AEEStdLib.h"           // Helper Function definitions
#include "Drawable.h"           // Drawable interface

class FlashText : public Drawable { // implement interface
private:
    IDisplay      *pIDisplay;
    const char    *pText;         // text to draw
    bool          redraw;        // toggle screen display
    bool          constructed;
    void          drawText();     // display text
    void          clear();        // clear screen
    AECHAR        buffer[50];
};
```

FlashText.h (continued)

public:

```
FlashText(const char *text, IDisplay *pd) {
    pText = text;  pIDisplay = pd;
    redraw = true; constructed = true;
}
bool objectBuilt() const { return constructed; }
const char *getText() const { return pText; }
void setText(const char *text) { pText = text; }
virtual void paint() {
    if (redraw)
        drawText();
    else
        clear();
    redraw = (redraw) ? false : true;
}
virtual const char *myType() const { return "FlashText"; }
};
```

FlashText.h (continued)

```
inline void FlashText::drawText() {
    IDISPLAY_ClearScreen(pIDisplay);
    IDISPLAY_DrawText(pIDisplay, AEE_FONT_BOLD,
        STRTOWSTR(pText, buffer, sizeof(buffer)),
        -1, 0, 0, NULL,
        IDF_ALIGN_CENTER | IDF_ALIGN_MIDDLE);
    IDISPLAY_Update(pIDisplay);
}

inline void FlashText::clear() {
    IDISPLAY_ClearScreen(pIDisplay);
    IDISPLAY_Update(pIDisplay);
}
#endif
```

DisplayTimer Example

- **Creating Objects**

```
pFlashText = new FlashText("Stay Awhile", pIDisplay);  
pDrawTimer = new DrawTimer(200, pFlashText, pIShell);  
pSoundPlayer = new SoundPlayer("StayAwhile.mp3", pIShell);
```

- **Event Handling**

```
case EVT_APP_START:  
    pDrawTimer->start();           // start Flashing Text  
    pSoundPlayer->start();         // play MP3  
    return true;  
  
case EVT_APP_STOP:  
    pDrawTimer->stop();            // stop Flashing Text  
    pSoundPlayer->stop();         // stop SoundPlayer  
    return true;
```

Templates with BREW

- **Why Use Templates?**
 - Type independent
 - Resolved at compile-time
 - Good for libraries
- **Template Functions**
 - Implements generic algorithms
 - Reusable
- **Template Classes**
 - Implements generic containers
 - Helps with polymorphism

Template Function Overloading

- **Max of two items**

```
int m = 34, n = 56;
int imax = max(m, n);           // 56
String s = "fast", t = "fist";
String smax = max(s, t);       // fist
```

- **Max item in array**

```
int buf1[] = { 34, -12, 88, 13, 44 };
int imax = max(buf1, 5);       // 88
String buf2[] = { "foot", "fast", "fate", "fist" };
String maxs = max(buf2, 4);    // foot
```

Template Functions

- **Max of two items**

```
template <class TYPE>
inline const TYPE & max(const TYPE & a, const TYPE & b) {
    return a > b ? a : b;
}
```

- **Max item in array**

```
template <class TYPE>
inline TYPE max(const TYPE *array, int length) {
    TYPE maxv = array[0];
    for (int i = 1; i < length; i++)
        if (array[i] > maxv)
            maxv = array[i];
    return maxv;
}
```

Template Functions

- **Specializing for C char arrays**

```
const char *buf3[] = { "foot", "fast", "fate", "fist" };  
const char *ps = max(buf3, 4);    // foot
```

- **Max C string in array**

```
inline const char *max(const char **array, int length) {  
    const char *maxv = array[0];  
    for (int i = 1; i < length; i++)  
        if (STRCMP(array[i], maxv) > 0)  
            maxv = array[i];  
    return maxv;  
}
```

Template Classes

- **Generic Stack**

```
Stack<int> s(10);
for (int i = 0; i < 10; i++)
    if (!s.full())                // stack not full?
        s.push(i);              // push item on stack
Stack<int> t = s;                 // copy stack
Stack<int> w;                    // default stack (length 80)
w = s;                          // stack assignment
int newlen = w.length();        // length of stack
for (; !s.empty(); s.pop()) {   // stack not empty?
    int item = s.top();         // get data from stack
    // do something with item...
}
Stack<String> b(10);            // Stack of C++ strings
```

Template Stack Class

```
#ifndef __STACK_H__
#define __STACK_H__
// Stack.h - header file for object
#include "AEEStdLib.h"      // Helper Function definitions
#include "Object.h"        // Object superclass

template <class TYPE>
class Stack : public Object {
private:
    TYPE *s;
    int len;
    int count;
public:
    Stack(int length = 80) {
        s = new TYPE[len = length];
        count = 0;
    }
}
```

Template Class Stack (continued)

```
Stack(const Stack<TYPE> & s1);          // copy constructor
Stack<TYPE> & operator=(const Stack<TYPE> & s1); // assign
virtual ~Stack() { delete [] s; }      // destructor
virtual const char *myType() const { return "Stack"; }

bool empty() const { return count == 0; }
bool full() const { return count == len; }
int length() const { return len; }
int nitems() const { return count; }

void push(const TYPE & item)
    { if (!full()) s[count++] = item; }
void pop() { if (!empty()) --count; }
const TYPE & top() const { return s[count-1]; }
};
```

Template Class Stack (continued)

```
template <class TYPE>
inline Stack<TYPE>::Stack(const Stack<TYPE> & s1) {
    s = new TYPE[len = s1.len];
    count = s1.count;
    for (int i = 0; i < len; i++)
        s[i] = s1.s[i];
}
template <class TYPE> inline
Stack<TYPE> & Stack<TYPE>::operator=(const Stack<TYPE> & s1) {
    TYPE *ns = new TYPE[len = s1.len];
    for (int i = 0; i < len; i++)
        ns[i] = s1.s[i];
    delete [] s;
    s = ns;
    count = s1.count;
    return *this;
}
#endif
```

Stack of Objects

```
class MyApp : public AEEApplet {
private:
    Stack<Object *> *pStack;
    FlashText      *pFlashText;
    SoundPlayer    *pSoundPlayer;
    DrawTimer      *pDrawTimer;
    . . .
};

bool MyApp::OnAppInitData() {
    pStack = new Stack<Object *>(10);
    pFlashText = new FlashText("Stay Awhile", pIDisplay);
    pStack->push(new DrawTimer(200, pFlashText, pIShell));
    pStack->push(new SoundPlayer("StayAwhile.mp3", pIShell));
    . . .
}
```

Stack of Objects

```
void MyApp::OnPlayMP3() {
    Object *pObject = pStack->top();
    if (pObject->myType() == "SoundPlayer") {
        pSoundPlayer = (SoundPlayer *)pObject;
        pSoundPlayer->start();    // play MP3
        . . .
    }
}

void MyApp::OnFlashingText() {
    Object *pObject = pStack->top();
    if (pObject->myType() == "DrawTimer") {
        pDrawTimer = (DrawTimer *)pObject;
        pDrawTimer->start();    // start Flashing Text
        . . .
    }
}
```

Stack of Objects

```
bool MyApp::OnAppFreeData () {  
    for (; !pStack->empty(); pStack->pop())  
        delete pStack->top();        // delete Object  
    delete pStack;                    // delete Stack Object  
    delete pFlashText;                // delete FlashText Object  
}
```

C++ with BREW

- **All programs compiled with ARM v1.2**
- **Benefits**
 - C++ has better type checking than C
 - Data abstraction and encapsulation
 - Good for OOP and reusability
- **Classes and Interfaces**
 - Encapsulate access to BREW APIs
 - Define controlled interface for BREW APIs
- **Constructors and Destructors**
 - Constructors allocate BREW resources
 - Destructor release BREW resources

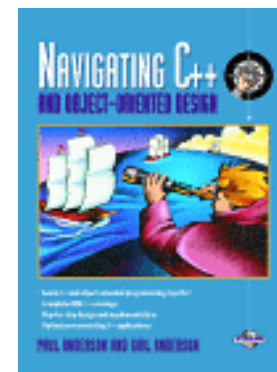
C++ with BREW

- **Class Member Functions**
 - Defines interface to BREW API calls
 - Hides internal complexities of BREW API
- **Best Practices**
 - Use inline functions for performance
 - Use virtual functions for polymorphism
 - Use inheritance and containment for reusability
 - Use interfaces to specify promised behaviors
- **Pros and Cons**
 - C++ objects are reusable
 - Overhead in execution time and code space

The End

- **Contact Info**

- Anderson Software Group, Inc.
- Voice: 760.436.9163
- Email: paul.anderson@asgteach.com
- Website: www.asgteach.com
- C++ Textbook
- BREW Training



- **Questions/Answers**

