Using Microsoft Foundation Classes 2.0 with Borland C++ 3.1

Prepared by:
Borland International
1800 Green Hills Road
PO Box 660001

Scotts Valley, CA 95067-0001

Table of Contents

Overview

This document describes how to build and use Microsoft Foundation Class 2.0 (MFC 2.0) programs with Borland C++ 3.1. This document is divided into several sections. The first describes changes to make to the MFC source files, the second discusses building the MFC libraries, and the final section explores compiling MFC 2.0 example programs.

The changes described in this document are necessary for several reasons. Some relate to non-standard syntax that Microsoft uses in MFC, while others relate to differences in assembly language extensions and makefile conventions.

If you are using MFC for your Windows programs, you will find many advantages in using Borland C++ 3.1 instead of Microsoft Visual C++:

+ Borland provides far faster development times, with its lightning fast compiles.
+ Borland provides a far more robust, compliant implementation of C++.
+ Borland provides templates and other advanced C++ features.
+ Borland provides automatic, safe precompiled headers and a smart project manager.

You may also be interested in information about ObjectWindows 2.0. ObjectWindows 2.0 provides many significant enhancements to ObjectWindows 1.0: 16 to 32 bit portability; broad API coverage, including GDI encapsulation; diagnostics; high level controls such as toolbars, status lines, sliders, and gauges; document/view support; full C++ exception support; and much more. To get an advance copy of ObjectWindows 2.0, sign up for the Borland C++ for Win32 Early Experience Program.

Changes to MFC

This section describes the various changes you will need to make to the MFC 2.0 source code. Note that the changes mentioned are based on the version of MFC 2.0 in the first shipping packages of Visual C++ (files dated on 02/08/93 at 01:01). The modifications are subject to change should Microsoft provide interim versions of MFC 2.0.

To use MFC 2.0 with Borland C++ 3.1 you must modify the following files:

OBJCORE.CPP
AFX.INL
AFXCOLL.H
AFXOLE.H
AFXPRIV.H
AFXDLGS.H
AFXEXT.H
AFXWIN.H
AFX.H
AFXPEN.H

All modifications applied to the MFC files will be enclosed by preprocessor statements verifying that the macro __BORLANDC__ is defined. This will ensure that the changes are only in effect when the sources are compiled with the Borland C++ compiler.

The bulk of the modifications will be performed by a new header file: AFX4BC.H. This header file makes use of the preprocessor to define and redefine various macros and also creates two templates for the function min() and max(). The contents of AFX4BC.H follows:

AFX4BC.H

```
#if     !defined(__AFX4BC_H)
#define          __AFX4BC_H

#if     defined(__BORLANDC__)

#if     defined(__SMALL__)    ||  defined(__MEDIUM__)
#define  _NEARDATA
#include <dos.h>
#endif  //__SMALL__   ||   __MEDIUM__

#define  _WINDOWS
#define  _PORTABLE
```

```
#define  BASED_CODE
#define  BASED_DEBUG
#define  BASED_STACK
#define  AFX_STACK_DATA
#define  AFXAPI_DATA
#define  AFXAPI_DATA_TYPE
#define  AFX_MSG_CALL
#define  NO_VBX_SUPPORT


#if     !defined(__STAT_H)
#include <sys\stat.h>
#endif  //__STAT_H

#define  __MSC
#define  _access     access
#define  _find_t     find_t
#define  _gcvt       gcvt
#define  _itoa       itoa
#define  _lseek      lseek
#define  _locking    locking
#define  _LK_UNLCK   LK_UNLCK
#define  _LK_NBLCK   LK_NBLCK

#define  _diskfree_t diskfree_t
#define  _stat       stat
#define  _fstat(h,b) fstat((h), (b))


template <class T> T min( T t1, T t2 ) {  return t1 > t2 ?  t2 : t1;  };
template <class T> T max( T t1, T t2 ) {  return t1 > t2 ?  t1 : t2;  };


#endif  //  __BORLANDC__
#endif  //  __AFX4BC_H
```

Modifying OBJCORE.CPP

There are several changes to OBJCORE.CPP. The function AfxAssertValidObject() must be modified to accomodate the near vptrs used by Borland C++ by default. If you plan to use far vtables always, then the ASSERT() statement should probably be left uncommented. By default, however, Borland C++ uses near vtables.

```
// Diagnostic Support

#ifdef _DEBUG
extern "C" void AFXAPI AfxAssertValidObject(const CObject* pOb,
                        LPCSTR lpszFileName, int nLine)
```

```
{
      if (pOb == NULL)
      {
            TRACE0("ASSERT_VALID fails with NULL pointer\n");
            AfxAssertFailedLine(lpszFileName, nLine);
            return;     // quick escape
      }
      if (!AfxIsValidAddress(pOb, sizeof(CObject)))
      {
            TRACE0("ASSERT_VALID fails with illegal pointer\n");
            AfxAssertFailedLine(lpszFileName, nLine);
            return;     // quick escape
      }

#ifndef _M_I86SM
      // check to make sure the far VTable pointer is valid
#if !defined(__BORLANDC__)
      ASSERT(sizeof(CObject) == sizeof(void FAR*));
#endif
```

The next modification to OBJCORE.CPP is required to avoid ambiguity in near data memory models. The modification is not required if you plan to use only the Compact or Large memory models. The function CRuntimeClass::ConstructObject() is modified as follows:

```
BOOL CRuntimeClass::ConstructObject(void* pThis)
// dynamically construct an instance of this class in the memory pointed
//    to by 'pThis'.  Return FALSE if can't construct (i.e. an abstract
class)
{
#if defined(__BORLANDC__)
   ASSERT(AfxIsValidAddress((const void FAR*)pThis,
                              m_nObjectSize));
#else
   ASSERT(AfxIsValidAddress(pThis, m_nObjectSize));
#endif
```

The final modification required in OBJCORE.CPP is in the function AfxInitialize(). The modification adds a cast since *AfxNewHandler()* does not match the parameter type of *_AfxSetNewHandler().*

```
// hook in our own new_handler
static BOOL AfxInitialize()
{
      (void)_afx_version();
#ifdef _DEBUG
      // Force reference of the following symbols for CodeView
```

```
      (void)afxTraceEnabled;
      (void)afxMemDF;
#ifdef _WINDOWS
      (void)afxTraceFlags;
#endif
      if (!AfxDiagnosticInit())
            return FALSE;
#endif //_DEBUG

#if defined(__BORLANDC__)
    _AfxSetNewHandler( ( void (*)() )AfxNewHandler );
#else
      _AfxSetNewHandler(AfxNewHandler);
#endif
      return TRUE;
}
```

Modifying AFX.INL

Because Borland C++ does not support the '_segname' modifier, modify the function _AfxGetPtrFromFarPtr() as follows. NOTE: The code no longer verifies that the segment of the far pointer points to _DATA. A similar check could be implemented by verifying the segment against the segment of a variable known to be in the _DATA segment.

```
// Inline helpers for implementation

_AFX_INLINE void* PASCAL _AfxGetPtrFromFarPtr(void FAR* lp)
{
#ifdef _NEARDATA
#if defined(__BORLANDC__)
   return(void*)_AFX_FP_OFF(lp);
#else
   ASSERT(_AFX_FP_SEG(lp) == _segname("_DATA"));
   return (void*)_AFX_FP_OFF(lp);
#endif
#else
   return lp;
#endif
}
```

Modifying AFX.H

As mentioned earlier, the bulk of the modifications required can be performed by taking advantage of the C preprocessor. The file AFX4BC.H containing the new definitions or redefinitions must be visible to all framework sources. A good place to insert it is in AFX.H. An include statement is inserted near the top of this file as

illustrated in the following:

```
#ifndef __AFX_H__
#define __AFX_H__

#ifndef __cplusplus
#error Microsoft Foundation Classes require C++ compilation (use a .cpp
suffix)
#endif

/////////////////////////////////////////////////////////////////////////
#if   defined(__BORLANDC__)
#include "afx4bc.h"
#endif

#include "afxver_.h"        // Target version control
```

Modifying Miscellaneous .H files

The following recommended changes for the files AFXCOLL.H, AFXOLE.H, AFXPRIV.H, AFXDLGS.H, AFXEXT.H, AFXWIN.H, AFXPEN.H AND AFX.H are not strictly required to use the Foundation Classes with Borland C++. They will, however, allow more flexibility when configuring the compiler. For example, if you wish to use far virtual tables to free up DGROUP space or to use memory models other than LARGE, then the following changes are required. The aforementioned headers all contain the following statements at the end of the file:

```
#undef AFXAPP_DATA
#define AFXAPP_DATA     NEAR
```

While MFC 2.0 uses various macros instead of hardcoding modifiers such as NEAR or FAR, the statements given above seem to be an exception: the NEAR keyword is actually indirectly hardcoded. This section should therefore be commented out when compiling with Borland C++. Make the following changes to the end of afxcoll.h, afxole.h, afxpriv.h, afxdlgs.h, afxext.h, afxwin.h, afxpen.h and afx.h:

```
#if   !defined(__BORLANDC__)
#undef AFXAPP_DATA
#define AFXAPP_DATA     NEAR
#endif
```

Building the MFC 2.0 Library

The easiest way to build the MFC 2.0 library with the Borland C++ compiler is by using a makefile. The following makefile was used to build the large memory model library. You may need to edit some of the macros (or define them as environment variables) prior to running make. The most important macros are:

MFC_LIBNAME — The name of the libraries to build. The default name is BCMFC_*x* where *x* is the memory model. (NOTE: due to size of the library, this makefile actually creates three libraries named MFC_LIBNAME# where # is 1, 2 and 3)

MODEL — Memory model (e.g. 'l' for large or 'c' for compact)

(NOTE: while it's possible to create NEAR CODE versions of the library, any involved project will most likely result in an overflow of the _TEXT segment, the only CODE segment in the small and compact memory models).

BCPATH — The root directory of Borland C++ (defaults to C:\BORLANDC).

MFCPATH — The root directory of MFC (defaults to C:\MSVC\MFC).

For example, to build the large memory model (default) you can use the following command [we'll assume that your copy of Borland C++ is located on the D: drive in a directory named BC31] after copying the makefile to the MFC\SRC directory:

```
MAKE -DBCPATH=D:\BC31 -f BCMFC.MAK
```

This above will place the newly created libraries in the MFC\LIB directory and placed the .OBJ files in a new directory, MFC\SRC\OBJS_x where x is the memory model.

A copy of the makefile is included below:

BCMFC.MAK

```
.AUTODEPEND

#
# Uncomment and define the following macros if your setup differs
# from the defaults ( C:\BORLANDC and C:\MSVC\MFC for BCPATH and
# MFCPATH respectively )
#
#BCPATH=
#MFCPATH=
#


!if !$d(MODEL)                  #Memory Model of Library & Example
MODEL=l
!endif

!if !$d(MFC_LIBNAME)            #Base name of MFC library built using
BC++
MFC_LIBNAME=BCMFC_$(MODEL)
!endif

!if !$d(BCPATH)                 #Root directory of Borland C++
BCPATH=C:\BORLANDC
!endif

!if !$d(MFCPATH)                #Root directory of MFC
MFCPATH=C:\MSVC\MFC
!endif

!if !$d(INCPATH)                #Include paths for compiler
INCPATH=$(BCPATH)\INCLUDE;$(MFCPATH)\INCLUDE
!endif

!if !$d(OBJDIR)                 #Directory for .OBJs
OBJDIR=.\OBJS_$(MODEL)
!endif

!if !$d(LIBDIR)                 #Directory for .LIBs
LIBDIR=$(MFCPATH)\LIB
!endif

!if !$d(LIBFLAGS)
LIBFLAGS = /0 /P128
!endif


!if !$d(NODEBUG)
```

```
DBGFLAGS=-D_DEBUG -v
!else
DBGFLAGS=-v-
!endif

!if $d(OBJDIR)
O_DIR=-n$(OBJDIR)
!endif

!if !$d(CFLAGS)
CFLAGS = -c -w-hid -m$(MODEL) $(O_DIR) -WS \
         $(DBGFLAGS) -G -H=$(OBJDIR)\$(MFC_LIBNAME).sym
!endif

TLIB   =  TLIB
CC     =  BCC +$(OBJDIR)\$(MFC_LIBNAME).cfg


.PATH.OBJ=$(OBJDIR)

.cpp.obj:
   $(CC) -m$(MODEL) {$< }


OBJ_LIST1= afxabort.obj     \
          afxasert.obj     \
          afxinl1.obj      \
          afxinl2.obj      \
          afxmem.obj       \
          afxtrace.obj     \
          afxver.obj       \
          appcore.obj      \
          appdlg.obj       \
          appgray.obj      \
          apphelp.obj      \
          apphelpx.obj     \
          appprnt.obj      \
          appui.obj        \
          arccore.obj      \
          arcex.obj        \
          arcobj.obj       \
          array_b.obj      \
          array_d.obj      \
          array_o.obj      \
          array_p.obj      \
          array_s.obj      \
          array_u.obj      \
          array_w.obj      \
          auxdata.obj      \
          barcore.obj      \
          bardlg.obj       \
          bartool.obj      \
          cmdtarg.obj
```

1
1

```
OBJ_LIST2= dcmeta.obj       \
          dcprev.obj       \
          dlgclr.obj       \
          dlgcore.obj      \
          dlgdata.obj      \
          dlgfile.obj      \
          dlgfloat.obj     \
          dlgfnt.obj       \
          dlgfr.obj        \
          dlgprnt.obj      \
          doccore.obj      \
          docmulti.obj     \
          docsingl.obj     \
          doctempl.obj     \
          dumpcont.obj     \
          dumpflt.obj      \
          dumpinit.obj     \
          dumpout.obj      \
          except.obj       \
          filecore.obj     \
          filemem.obj      \
          filest.obj       \
          filetxt.obj      \
          filex.obj        \
          list_o.obj       \
          list_p.obj       \
          list_s.obj       \
          map_pp.obj       \
          map_pw.obj       \
          map_so.obj       \
          map_sp.obj       \
          map_ss.obj       \
          map_wo.obj       \
          map_wp.obj       \
          objcore.obj      \
          olecli.obj       \
          oledoc.obj       \
          olefile.obj      \
          olemisc.obj      \
          olesvr.obj

OBJ_LIST3= oletsvr.obj      \
          oleui.obj        \
          oleui2.obj       \
          penctrl.obj      \
          plex.obj         \
          strcore.obj      \
          strex.obj        \
          timecore.obj     \
          validadd.obj     \
          viewcore.obj     \
          viewedit.obj     \
```

```
          viewform.obj     \
          viewprev.obj     \
          viewprnt.obj     \
          viewscrl.obj     \
          winbtn.obj       \
          wincore.obj      \
          winctrl.obj      \
          winfrm.obj       \
          wingdi.obj       \
          winhand.obj      \
          winmain.obj      \
          winmdi.obj       \
          winmenu.obj      \
          winsplit.obj     \
          winstr.obj


ALL: $(OBJDIR)\$(MFC_LIBNAME).CFG  \
    $(LIBDIR)\$(MFC_LIBNAME)1.LIB \
    $(LIBDIR)\$(MFC_LIBNAME)2.LIB \
    $(LIBDIR)\$(MFC_LIBNAME)3.LIB


$(LIBDIR)\$(MFC_LIBNAME)1.LIB : $(OBJ_LIST1)
   &TLIB $(LIBFLAGS) $< {-+$? }


$(LIBDIR)\$(MFC_LIBNAME)2.LIB : $(OBJ_LIST2)
   &TLIB $(LIBFLAGS) $< {-+$? }


$(LIBDIR)\$(MFC_LIBNAME)3.LIB : $(OBJ_LIST3)
   &TLIB $(LIBFLAGS) $< {-+$? }


$(OBJ_LIST1) $(OBJ_LIST2) $(OBJ_LIST3): $(OBJDIR)\$(MFC_LIBNAME).cfg


$(OBJDIR)\$(MFC_LIBNAME).cfg : BCMFC.MAK
   if not exist $(OBJDIR)\NUL md $(OBJDIR)
   echo   -I$(INCPATH)                    >  $<
   echo    $(CFLAGS)                      >> $<
```

Building the MFC Examples

You should be able to build the MFC 2.0 examples by simply creating a new makefile to work with Borland C++ 3.1. A prototypical makefile follows. The file can be used as a template for most of the examples shipping with MFC 2.0. The only items likely to change when using the makefile for the examples is the 'EXENAME' macro and the 'OBJS' macro. The former is the name of the target executable while the latter is a list of all object modules making up the example. The makefile expects a .DEF file and an .RC file with names matching the 'EXENAME'. Note that supplied .DEF files for the MFC examples do not utilize the STACKSIZE directive. A STACKSIZE statement should be added specifying a value of 10240.

To build the CHKBOOK example using Borland C++, simply copy the makefile to the MFC\SAMPLES\CHKBOOK directory and then execute the following command:

MAKE -f CHKBOOK.BC

[NOTE: We'll use the .BC extension for the Borland C++ makefiles to avoid conflict with the makefiles

MAKEFILE for the CHKBOOK MFC Sample

```
EXENAME     = CHKBOOK
OBJS        = $(EXENAME).OBJ BOOKVW.OBJ CHECKDOC.OBJ CHECKVW.OBJ \
              CHKBKFRM.OBJ DOLLCENT.OBJ FXRECDOC.OBJ MAINFRM.OBJ \
              ROWVIEW.OBJ STDAFX.OBJ


#
# Uncomment and define the following macros if your setup differs
# from the defaults ( C:\BORLANDC and C:\MSVC\MFC for BCPATH and
# MFCPATH respectively )
#
#BCPATH=
#MFCPATH=
#

# ======================================================================== #
#  The following builds the executable EXENAME [macro defined above].  #
#  The files 'EXENAME'.DEF and 'EXENAME'.RC must exist.                 #
#  Some of the MACROS used below may need to be defined above if your  #
#  your setup differs from the expected default.                        #
#                                                                       #
#    MACRO        DEFAULT                                              #
#   ------------ ------------------------------------------------------  #
#    MODEL        l           ( i.e.  Large memory model )             #
#    MFC_LIBNAME  BCMFC_x     ( where x is the model, eg. BCMFC_L      #
```

```
#    BCPATH       C:\BORLANDC ( root directory of Borland C++ )        #
#    MFCPATH      C:\MSVC\MFC ( root directory of MFC )                #
#                                                                      #
# ==================================================================== #

.AUTODEPEND

!if !$d(MODEL)                  #Memory Model of Library & Example
MODEL=l
!endif

!if !$d(MFC_LIBNAME)            #Name of MFC library built using BC++
MFC_LIBNAME=BCMFC_$(MODEL)
!endif

!if !$d(BCPATH)                 #Root directory of Borland C++
BCPATH=C:\BORLANDC
!endif

!if !$d(MFCPATH)                #Root directory of MFC
MFCPATH=C:\MSVC\MFC
!endif

!if !$d(LIBPATH)                #Paths for Libraries (MFC & BC)
LIBPATH=$(BCPATH)\LIB;$(MFCPATH)\LIB
!endif

!if !$d(INCPATH)                #Paths for Include files (MFC & BC)
INCPATH=$(BCPATH)\INCLUDE;$(MFCPATH)\INCLUDE
!endif

!if !$d(NODEBUG)
DBGFLAGS=-D_DEBUG -v
LNKDBG=/v+
!else
DBGFLAGS=-v-
LNKDBG=/v-
!endif

!if !$d(CFLAGS)
CFLAGS= -c -m$(MODEL) -w-hid -WS $(DBGFLAGS) -G -H=$(EXENAME).sym
!endif

LINK  = TLINK
CC    = BCC +$(EXENAME).CFG


.cpp.obj:
   $(CC) -m$(MODEL) {$< }


.rc.res:
   rc -r -i$(INCPATH) $<
```

```
$(EXENAME).exe: $(EXENAME).cfg $(OBJS) $(EXENAME).res $(EXENAME).def
   $(LINK) /Twe /L$(LIBPATH) $(LNKDBG) /Vt /c /s @&&|
c0w$(MODEL) $(OBJS)
$(EXENAME)
$(EXENAME)
/v- $(MFC_LIBNAME)1 $(MFC_LIBNAME)2 $(MFC_LIBNAME)3 mathw$(MODEL) +
import cw$(MODEL)
$(EXENAME).DEF
|
   rc -k $(EXENAME).res $(EXENAME).exe


$(OBJS) : $(EXENAME).cfg


$(EXENAME).cfg : $(EXENAME).bc
   echo   -I$(INCPATH)                    >  $.
   echo   -L$(LIBPATH)                    >> $.
   echo    $(CFLAGS)                      >> $.
```

MAKEFILE for the CTRLTEST Sample

To use the previous makefile for the CTRLTEST example, you need only change 'EXENAME' and 'OBJS' as shown in the following makefile:

```
EXENAME     = CTRLTEST
OBJS        = $(EXENAME).OBJ STDAFX.OBJ BBUTTON.OBJ CUSTLIST.OBJ \
              CUSTMENU.OBJ DERPEN.OBJ DERTEST.OBJ DLGPEN.OBJ     \
              FEATPEN.OBJ PAREDIT.OBJ PAREDIT2.OBJ SPIN.OBJ      \
              SPINTEST.OBJ SUBTEST.OBJ WCLSTEST.OBJ


#
# Uncomment and define the following macros if your setup differs
# from the defaults ( C:\BORLANDC and C:\MSVC\MFC for BCPATH and
# MFCPATH respectively )
#
#BCPATH=
#MFCPATH=
#

# ================================================================== #
#  The following builds the executable EXENAME [macro defined above].  #
#  The files 'EXENAME'.DEF and 'EXENAME'.RC must exist.                #
#  Some of the MACROS used below may need to be defined above if your  #
#  your setup differs from the expected default.                       #
#                                                                      #
```

```
#    MACRO        DEFAULT                                                  #
#   ------------ ---------------------------------------------------------  #
#    MODEL        l           ( i.e.  Large memory model )                 #
#    MFC_LIBNAME  BCMFC_x     ( where x is the model, eg. BCMFC_L          #
#    BCPATH       C:\BORLANDC ( root directory of Borland C++ )            #
#    MFCPATH      C:\MSVC\MFC ( root directory of MFC )                    #
#                                                                          #
# ======================================================================== #

.AUTODEPEND

!if !$d(MODEL)                  #Memory Model of Library & Example
MODEL=l
!endif

!if !$d(MFC_LIBNAME)            #Name of MFC library built using BC++
MFC_LIBNAME=BCMFC_$(MODEL)
!endif

!if !$d(BCPATH)                 #Root directory of Borland C++
BCPATH=C:\BORLANDC
!endif

!if !$d(MFCPATH)                #Root directory of MFC
MFCPATH=C:\MSVC\MFC
!endif

!if !$d(LIBPATH)                #Paths for Libraries (MFC & BC)
LIBPATH=$(BCPATH)\LIB;$(MFCPATH)\LIB
!endif

!if !$d(INCPATH)                #Paths for Include files (MFC & BC)
INCPATH=$(BCPATH)\INCLUDE;$(MFCPATH)\INCLUDE
!endif

!if !$d(NODEBUG)
DBGFLAGS=-D_DEBUG -v
LNKDBG=/v+
!else
DBGFLAGS=-v-
LNKDBG=/v-
!endif

!if !$d(CFLAGS)
CFLAGS= -c -m$(MODEL) -w-hid -WS $(DBGFLAGS) -G -H=$(EXENAME).sym
!endif

LINK  = TLINK
CC    = BCC +$(EXENAME).CFG


.cpp.obj:
   $(CC) -m$(MODEL) {$< }
```

```
.rc.res:
   rc -r -i$(INCPATH) $<


$(EXENAME).exe: $(EXENAME).cfg $(OBJS) $(EXENAME).res $(EXENAME).def
   $(LINK) /Twe /L$(LIBPATH) $(LNKDBG) /Vt /c /s @&&|
c0w$(MODEL) $(OBJS)
$(EXENAME)
$(EXENAME)
/v- $(MFC_LIBNAME)1 $(MFC_LIBNAME)2 $(MFC_LIBNAME)3 mathw$(MODEL) +
import cw$(MODEL)
$(EXENAME).DEF
|
   rc -k $(EXENAME).res $(EXENAME).exe


$(OBJS) : $(EXENAME).cfg


$(EXENAME).cfg : $(EXENAME).bc
   echo   -I$(INCPATH)                    >  $.
   echo   -L$(LIBPATH)                    >> $.
   echo    $(CFLAGS)                      >> $.
```

Additional Notes

The MFC libraries built with Borland C++ do not contain VBX support since the source code for the files providing this support are not provided with Visual C++. The files VBCORE.CPP and VBDATA.CPP are required for VBX support.

Building the DLL version of MFC with Borland C++ requires substantial and involved changes to the source code since the DLL specific classes assume VBX support but the sources to the latter have not been made available; (NOTE: The DLL specific sources do not check for the NO_VBX_SUPPORT macro) . Besides the VBX support requirement, a few other minor modifications will be required prior to building the DLL version using Borland C++:

- Borland C++ uses near vtables by default; to correctly share classes across DLL(s) and the application EXE requires that the classes be redefined with appropriate modifiers (i.e. huge and _export).
- The module AFXDLL.ASM hardcodes the non-portable Visual C++ mangling scheme;

The libraries are built with a '/0' passed to TLIB which purges all comment records, including debugging information. If you want to step through some of the MFC sources using Turbo Debugger for Windows, you can simply add the corresponding .OBJs to the 'OBJS' macro used in the example makefiles.

Some of the sample sources (e.g. several .CPP files in the CTRLTEST directory) have an additional semicolon after the 'DECLARE_MESSAGE_MAP()' macro. This results in a null statement within the class declaration [Null statements are not allowed within a class declaration according to the ARM section 17.5]. This can be remedied using the preprocessor. The following example uses the CTestApp class from CTRLTEST.CPP:

```
class CTestApp : public CWinApp
{
public:
   CTestApp();

   virtual BOOL InitInstance();
   //{{AFX_MSG(CTestApp)
   afx_msg void OnAppAbout();
   //}}AFX_MSG
#if !defined(__BORLANDC__)
     DECLARE_MESSAGE_MAP();
#else
      DECLARE_MESSAGE_MAP()
#endif
};
```