Welcome to C++Builder

Welcome to the C++Builder Field Test Refresh release! We thank you for taking the time to evaluate our latest tools for creating Windows programs using the C++ language.

We are distributing this pre-release Field Test Refresh so you can continue your product testing with the latest version of the software.

About this release

Although this Field Test Refresh is fully functional, please note that distributing applications generated with this Preview edition of C++Builder is prohibited.

Please remember that C++Builder is still in an unfinished state. As a precaution, you may consider backing up your important data files before programming with this tool.

Installation

To install C++Builder, run SETUP.EXE from the root directory of your distribution CD-ROM.

Refer to the file INSTALL.TXT, located on the root directory of your CD-ROM, for more information on installing this Field Test Refresh release.

Note: If you are using Windows NT version 4.0, you should make sure that you have build 1381 with Service Pack 1 installed (the blue screen shown during machine booting should show `Build 1381 Service Pack 1`). The service pack for this version of NT is available from Microsoft at:

http://www.microsoft.com/NTWksSupport

Product features

This section contains information on the following subjects:

- C++Builder interface and components
- Examples
- Help files
- Differences between C++Builder and Delphi
- C++ compiler modifications
- Object Pascal compiler modifications

C++Builder interface and components

C++Builder provides a rapid application development (RAD) environment and tools to help you create 32-bit Windows applications. It combines the visual development environment of Delphi with cached, pre-compiled headers and incremental/smart linking to provide fast-turnaround cycles while developing projects. C++Builder also provides a visual development environment with Delphi-level database connectivity support for C++.

C++Builder's design incorporates most Delphi interface and development components, but generates C++ code for projects, forms and code modules. The C++Builder BCC32.EXE compiler compiles C++ code accepted by BCC32 in Borland C++ 5.0x, and the DCC32.EXE compiler similarly compiles Object Pascal code from Delphi.

C++Builder uses the same VCL components as Delphi to support controls on the Component palette.

Examples

C++Builder comes with a multitude of example programs to help you get started. Use the File|Open Project command to open any of the .MAK project files located under the Examples directory, then press F9 to build and run the example. Here is a partial list of the examples you will find:

- FishFind.mak
- Switch.mak
- Threads.mak
- Web.mak
- ImagView.mak
- Contacts.mak
- TwoForms.mak
- ProcView.mak
- AutoSrv.mak
- AutoCon.mak

----> Note that for AutoCon.mak to build and run correctly, you must first build and run the "AutoSrv" example.

Help files

The Help file BCBPG.HLP contains the Programmers Guide material for C++Builder. This has been completely revised and updated with new material specific to your C++Builder package. Review this Help file for details on the C++ language extensions, exception handling, and other new features of the compiler. You will also find Component Writer's Guide material in the BCBCWG.HLP Help file. These Help files, and others, can be found in the Help directory off your main C++Builder directory.

Differences between C++Builder and Delphi

In C++Builder, project options are kept in project "makefiles," which end with a .MAK file extension. All compiler/linker options for a C++Builder project are stored in this fully functioning makefile that can be run outside C++Builder using the supplied command line tools. The project makefiles can be edited to change or add advanced compiler/linker options not supported by the UI; however, if you want to continue to use the project file with C++Builder, you must conform to the C++Builder makefile syntax.

The C++Builder Component Library is based on object file DLLs rather than the Delphi 2.0 .DCU file DLLs. With C++Builder, you can mix .pas and .cpp units in the same .DLL.

C++ compiler modifications

Native properties

Native properties have been added. An example:

```
_ _property String FileName =
  {read=GetFileName, write=SetFileName};
// ...
FileName = FileName + ".cpp";
```

Please note that to improve the readability of this document, a space has been added between all double-underbars.

Native closures

Native closures are supported. An example:

```
void (_ _closure *OnClickButton) (TObject* Sender);
// ...
OnClickButton( foo );  // call handler
OnClickButton = bar->MyHandleButton;  // set new handler
OnClickButton( foo );  // call different handler
```

See the minicomp example for usage of properties and closures. In addition, refer to the BCBCWG.HLP Help file for more information on properties and events (closures).

Keywords

_ _declspec(delphiclass | delphireturn)

_ _automated

_ _published

_ _closure

_ _property

_ _classid(class)

_ _dispid(int)

Other C++ additions

The following new classes have been added: AnsiString, Variant, ShortString, Currency, TDateTime, and Set.

The C++Builder runtime library provides functions which let you develop locale-sensitive applications. To support locale-sensitive applications, the RTL now contains functions that have wchar_t and multi-byte arguments. See the BCBRTL.HLP online Help files for the RTL details.

The following new open array helper macros have been added:

- OPENARRAY
- ARRAYOFCONST
- EXISTINGARRAY
- SLICE

The following new switches have been added to BCC32.EXE:

- **-Vx** switch for truly empty (0 length) structs.
- -**He** switch (extern types in .objs)
- **-Hs** switch (smart cached headers)

The following new #pragmas have been added to BCC32.EXE:

- #pragma link "obj" (object file dependencies)
- #pragma resource "res" (resource file dependencies)
- #pragma anon_struct on (for support of VCL variant records)

Support for dynamic_cast<> of VCL objects has been added.

Support for calling and defining dynamic and message functions has been added.

Support for OLE automation controllers with variant dispatching.

Object Pascal compiler modifications

The following new switches have been added to the Object Pascal compiler:

- **-jp** switch: creates Borland C++ compatible .obj files.
- **-jph** switch: creates C++Builder compatible header (.hpp) files from Object Pascal unit files (.dcl).
- **-jphn** switch: uses the Object Pascal unit name as the enclosing C++ namespace for both .objs and .hpps that are generated.
- **-n** switch: specify .dcu output directory
- **-nh** switch: specify .hpp output directory
- **-no** switch: specify .obj output directory

Old-style Object, Real, and Comp types are no longer supported in the Interface section of a Object Pascal unit. Although Comp is semi-supported as a non-numeric type, it should not be used; use Currency instead.

Known problems

The following list details the known problems in this Preview edition of C++Builder. These problems are being actively worked on and will be fixed in the final release of this product.

- There is a known problem where you might experience a system crash if you run out of disk space while using the C++Builder IDE. We suggest you have at least 10 Mb of available disk space on your working drive.
- Save As file operations are supported as long as the resulting file name for units is <modulename>.CPP. The .cpp extension is currently required and the <modulename> must meet the rules for a C++Builder name (1-32 alphanumeric characters consisting of letters, digits, and underscores).
- The inline assembler is not currently supported within the C++Builder IDE; you cannot use the ASM keyword within C++Builder projects. Because of ths
- If you encounter the error “Fatal: Unable to open file <Path>\lib\VCL.#00” when linking a project, use Project|Build All to regenerate all project files.
- It is recommended that you keep the Break on Exception check box checked for debugging purposes. This is checked by default in the Debugging section of the Preferences page of the Environment Options dialog.
- It has been reported that you might not be able to continue running your program after an exception is thrown while debugging.
- You might encounter a <cannot evaluate> error when you attempt to inspect or evaluate certain variables in the debugger.
- It has been reported that the debugger does not get properly notified of stack-fault exceptions when an application is linked with ILINK.
- It has been reported that after catching a hardware exception with the VCL’s hardware exception classes, that the debugger views do not contain valid data.
- If you are installing or removing components, make sure the Search Path on the Install Component dialog shows fully qualified paths; $(PRONTO) does not currently expand correctly.
- In this release, you cannot reliably export templates from a .DLL. If you need template definitions from existing engine code, you will have to link them statically.
- If you are rebuilding the Visual Component Library, you must delete the *.il? files between each library build. If you do not delete the *.il? files, the newly generated VCL will corrupt C++Builder when you try to load it. In addition, you must use ILINK whenever you rebuild the VCL.
- You might run into problems assigning values to local variables in catch blocks. This can happen when a local variable is used after the catch block is exited. If you must do this, a work around is to make the local variable static.
- TThread does not correctly initialize the C++ RTL; it should be avoided in this Preview edition release.
- The Code editor does not correctly search for MBCS characters.
- Currently, the default project always uses the **-o** option in the cpp->obj rule. This obstructs settings made in the output directory.
- If you place code or declarations in the project source file (WinMain) directly after the last USEXXX() macro line, it may get lost when the file is saved. For this release, put your code after the dashed comment line after the USEXXX() macros.
- If you select the Optimize for Speed option on the C++ page of the Project Options dialog box, C++Builder incorrectly adds the **-OS** option to the project makefile. To work around this, choose View|Project Makefile, then change the **-OS** option setting in CFLAG1 to **-O-S**.