Welcome

This document describes C++Builder 3.0. It includes:

What's New in C++Builder 3.0

The following is a summary of the new features in this release.

- A new Project Manager
  A new Project Manager allows you to combine projects that work together into a Project Group. This allows you to organize and work on interdependent projects such as separate tiers in a multi-tiered application or DLLs and executables that work together.
- Packages
  C++Builder allows you to build packages to install custom components and to share code at runtime.
- Code insight and Code editor enhancements
  Code insight provides templates and tool-tip expression evaluation as you work in the Code editor.
- New project options
  C++Builder has added project options to provide version information, advanced compiler options, and support for Assembly language compilation.
- Linker enhancements
  C++Builder now provides incremental linking to improve the time it takes to link programs.
- Debugger enhancements
  For debugging your code, C++Builder now provides new ways to view your code and to set breakpoints if memory is written to a certain address. In addition, you can debug individual executables and DLLs rather than having to debug a program within a project.
- Component templates
  You can now create component templates which allow you to group components on a form, save their arrangement and default properties, and then reuse the template on other forms.
- C++ language enhancements
  C++Builder now supports dynamic functions. It also has new #pragmas and improved template generation semantics. In addition, this version of C++Builder has upgraded its STL to the current ANSI committee draft of the Standard C++ Library implemented by RogueWave.
- Visual Component Library enhancements
  The VCL has a few new objects and changes to existing objects to support new Windows GUI features, charts, and ActiveX.
- Database enhancements
  This version of C++Builder includes enhancements to the Borland Database Engine.
- Batch file projects
- New Console wizard
- Key to the documentation
  C++Builder has updated its online and printed documentation accompanying this release.

New Project Manager

A new Project Manager allows you to combine projects that work together into a single Project Group. This allows you to organize and work on interdependent projects such as separate tiers in a multi-tiered application or DLLs and executables that work together.

Within each project, the Project Manager allows you to perform the same project management tasks as in C++Builder 1.0, such as add and remove files from projects, set project options, and compile projects. With project groups, you can add and remove projects from the project group and compile all projects within the group at one time.

For details, see Using the Project Manager.

New file extensions

With the addition of project groups, C++Builder uses different file extensions:

Project file	Extenstion	Description
Project group	.BPG	Makefile that builds all projects included in this project group
Project	.BPR	Makefile that builds a project.
Package	.BPK	Makefile that builds a package.

Prior to this release, C++Builder used the file extension .MAK for makefiles that built C++Builder projects. C++Builder now uses the extension .BPR. C++Builder uses a different extension to distinguish C++Builder project files from make files used by other C++ compilers. Specifically, a C++Builder project file is a subset of the full MAK.EXE syntax and not any user-written makefile can be used as a C++Builder project. C++Builder projects need special defines andmust have certain parts in a specific order.

C++Builder now generates files with the following file types:

File Type	Extenstion	Description
Package library	.BPL	File containing a compiled package.
Import library	.BPI	File containing an import library for compiled packages.
ActiveX	.OCX	Compiled ActiveX control or automation object.

Packages

You can create special dynamic-link libraries called packages for both design-time and runtime benefit. You create packages at design-time to install your custom components in the IDE. At runtime, you would create a package to share code among applications. By compiling the code shared among your applications in a separate runtime library, the application executable files are much smaller. Also, with runtime packages, your applications compile faster because only code unique to the application is compiled with each build.

For more information, see:

About packages

Runtime packages

Custom packages

Design-time packages

Creating and editing packages

Compiling packages

Deploying packages

Code insight and Code editor enhancements

Code insight is an IDE enhancement that displays provides templates and tool-tip expression evaluation. These features are updated dynamically as you work in the Code editor. To configure, choose Tools|Environment Options and select Code Insight.

Code templates

Code templates are common programming statements that you can quickly insert into your code. Templates are available for common C++ syntactic structures such as “if ......else” and “while” statements and “try ... catch” blocks.

- To access code templates in the editor, press Ctrl+J.
- To add or edit templates, choose Tools|Environment Options and select the Code Insight tab.

Code templates are not multibyte-enabled.

Tool-tip expression evaluation

You can quickly view the value of a variable or property while an application is stopped in debug mode. To do so, place the mouse cursor over any variable or property name in the Code editor and the value of the variable or property is shown in a pop-up window. This feature works best if compiler Optimization is turned off; choose Project|Options and select the C++ tab.

More Code editor enhancements

This version includes more debugger support. The Code editor window now displays a gutter and gutter glyphs. To configure, choose Tools|Environment Options|Display.

The input method editor (IME) has added support for Asian-language environments. Comments and string values can now contain multibyte characters. C++ identifiers (names of variables, constants, etc.) cannot use multibyte characters.

Text searches are supported across entire projects, across specified directories, and across all open files. To specify files to search, choose Search|Find In Files, or choose Search|Find and click the Find In Files tab.

New project options

You can now set more project options to include:

- Version information
- Advanced compiler options
- C++ options
- TLIB options

Version information

With this version of C++Builder, you can include version information in your C++Builder projects.

Choose Project|Options|VersionInfo to set version information for your project. In addition to updating version information manually before recompiling a project, you can select the Auto-increment Build Number option. Version information appears on the Version page of the Windows Properties dialog for the project's .EXE file.

C++ options

Choose Project|Options|C++ to set options specific to the C++ language. Previously, the C++ page contained standard compiler options available for compiling C++Builder projects.

The compile options that were available on this page in C++Builder 1.0 are now available on the Compiler page.

Advanced compiler options

More compiler options are now available by choosing Project|Options |Advanced compiler.

TLIB options

C++Builder allows you to set options on static libraries. TLIB options are available by choosing Project|Options|TLIB.

Linker enhancements

C++Builder includes many improvements to the incremental linker (ILINK).

While C++Builder supports both the incremental linker and traditional linker (TLINK), use the incremental linker for new development. You *must* use the incremental linker when using C++Builder IDE. TLINK exists for backward-compatibility.

ILINK is the default linker setting for IDE, so your projects will automatically use ILINK when you build from the C++Builder IDE. For command-line invocation, in your makefile, change the macro LINKER= from,

```
TLINK
```

to

```
ILINK
```

You can now generate import libraries by using the -Gi linker switch.

To disable incremental linking, use the -Gn switch, which prevents generation of state files.

The following linker switches have been added to support packages.

Switch	**Purpose**
-Tpp	Builds the project as a package. Included by default in package makefiles.
-Gi	Saves the generated BPI file. Included by default in package makefiles.
-Gpr	Generates a runtime-only package.
-Gpd	Generates design-time-only package.
-Gl	Generates a .LIB file.
-D	Saves the specified description with the package.

If you omit the -Gpr and -Gpd options from the linker command line, the resulting package works at both runtime and design-time. If you add both options to the linker command line, the linker ignores all but the last option; for example, if you list -Gpr -Gpd, the linker builds only a design-time package because -Gpd was the last option listed on the command line.

Some linker options that were previously supported by TLINK are not currently supported in ILINK. Makefiles that include these options will link without generating any warnings, but they will not affect the target file.

The following options currently have no affect in C++Builder 3.0 linking:

/Enn, /Gd, /Go, /Gt, /n, /Oc, /OS, /o, and /P.

Debugger enhancements

C++Builder includes the following enhancements to the debugger.

Debugging an executable no longer requires a project

Prior to this release, you had to open a project to debug an executable file. Now, you can now specify a pathname to the executable. To do so, Run|Parameters and enter the path to the .EXE in the “Host application” edit box, then press the Load button to load the executable in the debugger.

This is particularly useful when debugging a DLL project.

Debugging from the command line

You can use the command-line switch -d or /d to tell the IDE that you want to debug a certain program. For example, to debug a program called foo.exe, type at a command prompt:

```
bcb -dfoo.exe
```

or

```
bcb /dfoo.exe
```

New event log view

You can also display an event log from the debugger. It shows process control messages, breakpoint messages, OutputDebugString messages, and Windows messages.

Right-click in Event log to clear the event log, save the event log to a text file, add a comment to the event log, and set options for the event log.

Setting debugger options for all debugger projects

You can now set the behavior of the debugger for all your projects by setting global debugger options on the Tools|Environment Options|Debugger page. The information set here is valid when debugging any C++Builder project. For more information on the Debugger page, select the page and press F1.

Component templates

Component templates are groups of components that you add to a form in a single operation. Templates allow you to configure components on one form, then save their arrangement and default properties on the Component palette for reuse on other forms.

You can create component templates that are made up of a number of components. After arranging components on a form, setting their properties, and writing code for them, you can save them as a component template. Later, by selecting the template from the Component palette, you can place the preconfigured components on a form in a single step; all associated properties and event-handling code are added to your project at the same time.

Once you place the template on a form, you can reposition the components independently, reset their properties, and create or modify event handlers for them just as if you had placed each component in a separate operation.

To create a component template, you use Component|Create Component Template.

To remove a component template from the Component palette,

- Choose Component|Configure Palette.

C++ language enhancements

C++Builder includes the following enhancements to the C++ language:

- Standard C++ library
- Dynamic functions
- New #pragma package
- New #pragma option push and option pop
- Changes to template generation semantics
- Support for try/__finally exception construct

Standard C++ library

C++ Builder 3.0 comes with the Standard C++ library from RogueWave. The Standard C++ Library is a large and comprehensive collection of classes and functions that is RogueWave's implementation of the standard C++ language as defined by the International Standards Organization (ISO) and the American National Standards Institute (ANSI).

C++Builder 3.0 is compliant with the ANSI standard . It does not support nested templates.

The ANSI/ISO Standard C++ Library includes the following parts:

- A large set of data structures and algorithms formerly known as the Standard Template Library (STL).
- An IOStream facility
- A locale facility
- A templatized string class
- A templatized class for representing complex numbers
- A uniform framework for describing the execution environment, through the use of a template class named numeric_limits and specializations for each fundamental data type
- Memory management features
- Language support features
- Exception handling features
- A valarray class optimized for handling numeric arrays

You may experience some problems recompiling existing C++ programs with the new Standard C++ library as some C++ constructs defined in the previous version of STL are no longer part of the ANSI standard.

For more information, see the Standard C++ Library documentation available on the C++Builder CD.

Changes to template generation semantics

Template generation semantics have changed. In Borland C++Builder 1.0, all out-of-line methods for all template instances used by a program were generated. In C++Builder, only the following are generated:

- Those methods which were actually used
- Virtual methods of an instance
- All methods of *explicitly* instantiated classes

The advantage of this new behavior is that it results in significantly smaller .OBJs, .LIBs and .EXEs, depending on how heavily you use templates.

Optionally, you can use the '-Ja' switch to generate all methods, (the same behavior as in Borland C++Builder 1.0).

You can also force all of the out-of-line methods of a template instance to be generated by using the explicit template instantiation syntax defined in the ANSI C++ draft. The syntax is:

template class *classname***<***template parameter***>;**

The following STL example directs the compiler to generate all out-of-line methods for the "list<char>" class, regardless of whether they are referenced by the user's code:

```
template class list<char>
```

You can also explicitly instantiate a single method, or a single static data member of a template class, which means that the method is generated to the .OBJ even though it is not used:

```
template void classname <template parameter>:: methodname ();
```

VCL enhancements

The following components or component groups, objects, properties, methods, or events have been enhanced or are new to C++Builder:

User interface design

- Button and Toolbar components (new)
- Graphics enhancements
- TDatetimePicker (new)
- TAnimate (new)
- TPageControl, TTabControl
- TSplitter (new)
- TOpenPictureDialog and TSavePictureDialog (new)
- Support for Font Setting Changes
- GetComCtlVersion global function (new)

Other components

- QuickReport components
- TThreadList (new)
- TRegistryIniFile and TMemIniFile for storing application settings(new)
- TStaticText (new)

COM/ActiveX

- TOleContainer

Button and toolbar components

See also

Two new visual components, TToolBar and TCoolBar, appear on the Win32 page of the Component palette. These components provide new options for configuring buttons and tool bars. Toolbars and cool bars offer improved control over the alignment of buttons at runtime.

The Flat property of TToolBar, if set to true, lets graphics show through from under the tool bar. The Flat property of TSpeedButton, if set to true, makes a button's border invisible until the user points to it with the mouse.

Note: To deploy an application that uses TCoolBar or TToolBar with the Flat property set to true, you must have the latest version of COMCTL32.DLL (version 4.70, dated 8/9/96, approximately 370Kb). This file is available from Microsoft; if you've installed Internet Explorer 3.0 or MS Office 97, you already have it.

For details, see:

Designing Toolbars and coolbars

TToolBar object

TCoolBar object

Graphics enhancements

The following changes and enhancements have been made to graphics support:

Thread-safe support	Now provided in graphics classes for multithreaded operations.
TCanvas	Has two new methods: Lock and Unlock. The Lock method grabs exclusive access to a drawing canvas, and Unlock releases it. Background threads should always lock a canvas before drawing to it. The VCL is updated to lock the canvas implicitly during normal VCL painting operations (Paint method and OnPaint events).
Bitmap objects	Have implicit thread locks only in the Assign operation. If you need to have multiple threads writing to the same bitmap object, you should lock the bitmap's canvas to synchronize access to the bitmap.
TBitmap	Has new properties: DIBMemory returns a pointer to the image pixel data in memory so that an image can be directly modified. TBitmapHandleType allows you to convert an internal bitmap handle from DIB to device format, or vice-versa. Some color formatting may be lost in the conversion.
TBitmap	Has been enhanced to use DIB sections instead of memory streams. This reduces system memory use by 50% and preserves the pixel format of the original file even through bitmap modifications.
New pixel formats	Added support for 15bpp, 16bpp, and 32bpp BMP pixel formats, and for optional color palettes in high-color BMP files. Palettes assigned to high-color bitmaps are stored in the BMP file. Dithering and halftone palettes are supported when the destination bpp is less than the source bpp.
Palette realization	Has been removed from bitmap-loading code. This eliminates screen flicker when loading forms with 256-color bitmaps, and improves load times for bitmaps and forms by a factor of ten or more in large applications.
TGraphic	Has two new properties, Transparent and Palette, to indicate palette and transparency. When Transparent is true, the image does not completely cover its rectangular area. The read-only property Palette is a handle to the color palette of the image. If the graphic does not use palettes, Palette is 0.
TImage	No longer assumes TBitmap's palette and transparency settings. Instead it uses TGraphic's generic Palette and Transparent properties. TImage->PictureChanged now initiates a palette realization, if necessary.
InitGraphics routine	Is no longer required and has been deleted from graphics.hpp.

TDateTimePicker

See also

A new visual component, TDateTimePicker, appears on the Win32 page of the Component palette. TDateTimePicker displays a list box for entering dates or times. To deploy TDateTimePicker, you must have the latest version of COMCTL32.DLL (version 4.70, dated 8/9/96, approximately 370Kb). This file is available from Microsoft; if you've installed Internet Explorer 3.0 or MS Office 97, you already have it.

TAnimate

See also

A new visual component, TAnimate, appears on the Win32 page of the Component palette. TAnimate plays Audio Visual Interleaved (AVI) clips. TAnimate works with uncompressed AVI files or AVI clips compressed using run-length encoding (RLE).

TAnimate can obtain its AVI clip from an AVI resource, an AVI file, or, if the application is running under Windows 95 or NT 4.0, from Shell32.dll.

For details, see:

Using animation componentsl

Tanimate object

TPageControl, TTabControl

See also

TPageControl and TTabControl have new properties:

HotTrack	If set to true, highlights the caption of each tab as the mouse pointer passes over it.
ScrollOpposite	Moves previous rows of tabs to the other side of the control when a tab in a later row is selected.
TabPosition	Determines whether the tabs are arranged at the top or the bottom of the control.

TSplitter

See also

A new visual component, TSplitter, appears on the Additional page of the Component palette. Splitters allow you to divide a form into arbitrary rectangular regions that can be resized by the user at runtime.

For details, see

TSplitter object

TOpenPictureDialog and TSavePictureDialog

See also

Two new Common Dialog components, TOpenPictureDialog and TSavePictureDialog, appear on the Dialogs palette page. These components work like TOpenDialog and TSaveDialog except that they include a “preview” region that displays the selected graphic file before opening or saving it.

For other information, see

Selecting files to open

Saving files

TOpenPictureDialog object

TSavePictureDialog object

GetComCtlVersion global function

See also

The VCL contains a new global function, GetComCtlVersion, that returns the version of the common Windows controls DLL, COMCTL32.DLL. Since the behavior of these Windows controls depends on the version of the DLL, and DLL can be updated independent of your applications, your applications may need to check which version of the DLL that the application is running.

For details, see:

GetComCtlVersion_function

Support for font setting changes

Screen, control, and application objects now have new properties to update fonts whenever font settings have changed. These properties allow applications to use the Windows 95 Display Properties settings.

TScreen	Has the IconFont property, which stores the icon font setting from Windows 95.
TApplication	Has the UpdateMetricSettings property. If UpdateMetricSettings is true (the default), it notifies top-level forms when any font settings change.
TControl	Has a new DesktopFont property. If DesktopFont is true, it resets the TControl Font property.

QuickReport components

The new QuickReport component set includes a wizard to create reports easily. Reports designed with Quick Report 1.0 are automatically converted at design time to version 2.0.

TThreadList

A new class, TThreadList, creates a thread-safe list. You can add or remove objects without explicit locks.

For details, see using threads.

TRegistryIniFile and TMemIniFile for storing application settings

See also

The VCL provides new objects to ease the migration from 16-bit applications, which store settings in an INI file, to 32-bit applications, which store settings in the system registry.

- TRegistryIniFile
- TMemIniFile

Note: While TRegistryIniFile and TMemIniFile are provided for migration from Windows 3.x applications, new application development should store and retrieve application settings by using the system registry provided with Windows 95/NT. C++Builder provides objects for handling the system registry, such as TRegistry.

TRegistryIniFile enables the handling of the registry as if it were a Windows 3.x INI file. Instead of processing an INI file, TRegistryIniFile reads from and writes to the system registry. This allows you to change existing applications from using INI files to using the system registry with few code changes.

By finding all references to TIniFile in an application’s source code, replacing them with TRegistryIniFile, and recompiling the application, a developer can update an application to use the system registry instead of INI files without having to code any new logic into the application.

Note: TRegistryIniFile provides essentially the same functionality as TRegIniFile that was provided in C++Builder 1.x, but TRegistryIniFile is more useful for migrating a 16-bit Windows 3.x application to Windows 95/NT.

TRegistryIniFile is derived from TCustomIniFile.

TMemIniFile

TMemIniFile is provided to improve the performance of storing and retrieving INI file information. Windows 95 caches INI file information in memory while Windows NT reads and writes to disk. TMemIniFile allows your application to create its own cache to buffer INI file information so that the application works the same for both Windows 95 and Windows NT environments.

Using TMemIniFile can improve your application’s performance significantly if it performs many read/write operations to and from an INI file.

TMemIniFile is derived from TCustomIniFile.

TOleContainer

See also

A new property, AllowActiveDoc, has been added to the TOleContainer object. Set AllowActiveDoc to true to allow insertion of ActiveDocs (DocObjects) into the OleContainer; the OleContainer can respond when an embedded object requests an ActiveDoc interface.

TStaticText

See also

A new visual component, TStaticText, appears on the Additional page of the Component palette. TStaticText is a read-only text component like Label, except that it includes a window handle, which is useful when the component’s accelerator key must belong to a windowed control.

StaticText can be used in ActiveX property pages to provide users with feedback on the current state of the application.

Database enhancements

C++Builder includes many database enhancements, including:

- New Borland Database Engine (BDE)

VCL components for database support

In addition, the following VCL components have been added or changed for database support:

- DataSet changes for blob caching
- TField

New Borland Database Engine (BDE)

See also

C++Builder includes new versions of the BDE 4.5 and SQL Links. Most files associated with the BDE are in \Program Files\Borland\C++Builder\BDE. Help for the BDE is in BDE32.HLP.

To configure the BDE, you can launch the following driver configuration utilities from within the Database Explorer:

New BDE features include the following.

FoxPro support The dBASE driver now includes support for FoxPro compressed index (.CDX) and BLOB (.FPT) files, letting you open and create FoxPro 2.0, 2.5, and 2.6 tables.

See details on new FoxPro support.

Microsoft Access support

If you have a version of the Microsoft JET engine (included with Microsoft Access and FoxPro) installed on your system, you can now use the BDE to open or create Microsoft Access tables using the MSACCESS driver.

Also, the Access driver now supports referential integrity.

To switch a cursor's Access locking protocol between pessimistic and optimistic, toggle the Boolean curPESSIMISTICLOCKS property with DbiSetProp.

Note: BCD (binary coded decimal) support is not yet implemented for this driver. Setting the BCD Enabled parameter to true has no effect.

See details on creating Microsoft Access tables.

ODBC 3.0 support The new BDE supports ODBC 3.0 drivers.

Multibyte character support

The new BDE is fully multibyte-enabled.

Data Dictionary features

The Data Dictionary can now store domain and table constraints. These can be propagated to the client using remote DataSets or enforced using BDE cursors.

Parameter binding support

BDE now supports parameter binding for BLOBs and strings longer than 255 characters. See the description of DbiQSetParams in BDE32.HLP for details.

Configurable BLOB caching

You can now use the Database Explorer to configure the BDE’s BLOB caching.

IDAPI functions support Access named queries

IDAPI functions such as DbiQExecProcDirect and DbiOpenSPParamList now support Access named queries. BDE treats these functions as stored procedures.

Set workgroup information file

You can now set the workgroup information file (Access SystemDB) on a per-database basis using the BDE Administrator.

Launch driver utilities to configure BDE

To launch ODBC Administrator for ODBC drives:

1 Click Databases at the top of the Databases tab.

2 Choose Object|ODBC Administrator.

To launch BDE Administrator for the BDE drivers:

1 Click Databases at the top of the Databases tab.

2 Choose Object|ODBC Administrator.

3 Choose Object|BDE Administrator. Online Help for the BDE Administrator (BDEADMIN.EXE) is in BDEADMIN.HLP.

Details on new FoxPro support

To create a FoxPro table with the dBASE driver:

1 Choose the BDE Administrator Configuration page.

2 Set the LEVEL driver configuration parameter to 25.

To create FoxPro tables using the BDE API, pass the optional parameter LEVEL with a value of 25 when using DbiCreateTable to create a dBASE table.

To add a FoxPro index, set curTABLELEVEL to 25 using DbiSetProp before calling DbiAddIndex. DbiSetProp returns an error if the table is already using dBASE index or BLOB files.

To see if a cursor is referencing a FoxPro table, retrieve CURProps using DbiGetCursorProps and check if CURProps.iTblLevel is equal to FOXLEVEL25.

Details on creating Microsoft Access tables

In the BDE API, use the constant szMSACCESS when creating Access tables or checking the table type. The following table lists new physical data types for MSACCESS and their BDE logical equivalents:

MSACCESS physical datatype	BDE logical equivalent
fdACCAUTOINC	fldINT32, fldstAUTOINC
fldACCBIT	fldBOOL
fldACCBYTE	fldUINT16
fldACCCHAR	fldZSTRING
fldACCDATETIME	fldTIMESTAMP
fldACCDOUBLE	fldFLOAT
fldACCFLOAT	fldFLOAT
fldACCLONG	fldINT32
fldACCLONGBINARY	fldBLOB, fldstACCOLEOBJ
fldACCLONGTEXT	fldBLOB, fldstMEMO
fldACCMONEY	fldFLOAT, fldstMONEY
fldACCSHORT	fldINT16
fldACCVARCHAR	fldZSTRING

DataSet changes

See also

A new public property has been added to TDataSet, CacheBlobs.

```
__property bool CacheBlobs;
```

Setting this to true turns on binary large object (BLOB) caching, which improves performance when scrolling through records with on-screen BLOB fields (for example, in a DBCtrlGrid). Additional memory is used to store the cache.

In preparation for extending the functionality of datasets, BDE-specific dependencies have been removed from TDataSet and put into a new object, TBDEDataSet. Some additional functionality is now found in TBDEDataSet instead of TDataSet. TDataSet itself contains three new methods:

`IsEmpty`	Checks for a dataset without records
`CompareBookmarks`	Compares two bookmarks and returns 0 if they are equal
`BookmarkValid`	Verifies that a bookmark is valid

These changes may affect how your C++Builder 1.0 applications compile in this version. For compatibility issues with previous releases of C++Builder, see Upgrading to C++Builder 3.0 from version 1.0.

Note: For dataset provider and InterBase tables, resolving a deleted or modified record requires a uniquely maintained index.

TField

A new class method, IsBlob, has been added to the Tfield object, IsBlob.
IsBlob returns true if the field is a BLOB field.

Batch file project

C++Builder now allows you to create a create a project that allows you to run batch files.

To create a new batch file target,

1 Choose File|New and select the Batch file icon off the New page.

  C++Builder creates a new project with no source code editor nor VCL.

2 Choose View|Project Manager, select the project, right click and choose Options. The Batch file options dialog box appears.

3 Choose the method for invoking the batch file commands:

  By choosing Run, C++Builder invokes the commands directly in a Windows shell. It handles only executable programs; it cannot handle such interpreter commands as dir or cd.

  By choosing Command-line interpreter, C++Builder invokes the command-line interpreter as specified. Typically, this is $(COMSPEC), which evaluates to the command-line interperter defined in the environment variable (such as windows\command.com or 4dos\4dos.com).

4 In the Commands edit box, type the commands you want to include in the batch file.

5 Click OK.

  C++Builder saves the file using the batch file extension, .BAT. It also adds code to the project group file (*project*.BPG).

Note: If you have specified that this file uses the command-line interpreter, C++Builder adds a line to the top of the file, “REM CommandInterpreter: $(COMSPEC),” to tell the IDE to start up the command-line interpreter upon invoking this batch file. Do not remove this line.

To load an existing batch file,

1 Choose File|Open.

2 In Files of type, select Batch file (*.bat) to display batch files.

3 Double-click the desired .BAT file to open it.

4 Right-click and choose options to choose the method for invoking he batch file commands:

  By choosing Run, C++Builder invokes the commands directly into a Windows shell. It handles only executable programs.

  By choosing Command-line interpreter, C++Builder invokes the command-line interpreter as specified (such as windows\command.com or 4dos\4dos.com).

Note: If your batch file requires a command-line interpreter, you must set this option before invoking the batch file in C++Builder.

New Console wizard

To customize the project, specify the desired project parameters below.

Choose New|Console wizard to open the Console wizard dialog box. You customize how C++Builder creates your project by choosing whether the project:

- Is either a console or Window GUI application
- Is either a .DLL or .EXE
- Includes the VCL

C++Builder creates a project including only those elements specified.

Key to the documentation

C++Builder provides you with Help on the following topics:

- Using C++Builder - Help on the Integrated Development Environment (IDE) and basic usage information
- Programming with C++Builder - Help on basic programming tasks using C++Builder
- Visual Component Library Reference - Reference material on all the objects and components, including examples, of the Visual Component Library (VCL).

Note Although the VCL Reference Help documents every object and component, the actual objects and components included in your copy of C++Builder depends upon the version you purchased

- Developing Database Applications - Information on writing database applications with C++Builder
- Developing Internet Applications - Information on writing internet applications with C++Builder
- Creating ActiveX Controls - Information on writing ActiveX controls with C++Builder
- Creating Custom Components - Information on writing custom components using C++Builder
- C Runtime Library Reference - Reference material on the Runtime Library (RTL)
- Command-line Tools - Help on the command line tools (such as, IDETOBPR.EXE and GREP.EXE) included with C++Builder

Also included on the C++Builder CD is information about the RogueWave implementation of the Standard C++ Library. This documentation is located in \CBuilder3\Help\.

The following printed manuals also may be included in your copy of C++Builder:

Teach Yourself C++Builder in 14 Days (all versions)

Teach Yourself C++Builder 3.0 in 14 Days is designed to provide information so that you can become familiar with the C++Builder user interface. It also describes the basic processes involved in creating C++ applications in this environment.

Developer's Guide (Professional and Client/Server versions only)

The *C++Builder Developer's Guide* provides details on how to use C++Builder to develop applications. It provides in-depth information on intermediate to advanced programming topics you need to create database applications, custom components, and Internet and intranet applications. It also describes how to work with COM and ActiveX. Formerly, this material was covered in separate manuals on database application development, component writing, and a general-purpose programmer's guide.

The following manual is available for purchase from Borland. For more information, see Borland Online (http://www.borland.com/).

Visual Component Library Reference

The *Visual Component Library Reference* provides information on the most commonly used visual components. The information is now organized by object with properties, methods, and events subordinated to their respective objects. This way, you can easily locate in which object a property, method, or event originated.

Additional software provided with C++Builder

In addition to C++Builder, the CD-ROM contains:

- InterBase 5.0
- MS Internet Explorer 4.02
- Netscape Navigator 3

InterBase

The Professional and Client/Server versions of C++Builder receive copies of the InterBase database server.

Note: Be sure to read the installation instructions accompanying the software for any installation requirements.

MS Internet Explorer

The latest version of Microsoft's Internet Explorer, version 4, is distributed on the CD-ROM.

Netscape Navigator

The latest version of Netscape Navigator, version 4.02, is distributed on the CD-ROM.

Online resources

You can get information from any of these online sources:

World Wide Web:	http://www.borland.com/
FTP:	Technical documents available by anonymous ftp via ftp.borland.com.
Listserv:	To subscribe to electronic newsletters, use the online form at http://www.borland.com/feedback/listserv.html. Or, for Borland's international listserver, http://www.borland.com/feedback/intlist.html.
TECHFAX	Technical documents available by fax at1-800-822-4269 (North America).

World Wide Web

Check Borland's C++Builder Web site regularly at http://www.borland.com/bcppbuilder. The C++Builder Product Team posts sample applications, white papers, competitive analyses, answers to frequently asked questions, updated software, and information about new and existing products.

Newsgroups

Borland Online hosts a variety of newsgroups where users can exchange information about Borland development tools and their use. For a complete list, see http://www.borland.com/newsgroups/.

Contacting Borland for support

Borland offers a range of support services for C++Builder and other tools. For information, see our World Wide Web site at http://www.borland.com/.

For assistance outside of North America, contact your local Borland representative. For a list of offices and distributors world-wide, see http://www.borland.com/bww/.

Documentation Updates

- Project management for packages
- Installing components and packages
- Uninstalling components and packages

Note

The C++Builder Help system often refers to packages such as Vcl30.bpi and Vcl30dbx.bpi. These should be Vcl35.bpi and Vcl35dbx.bpi. Substitute any reference to packages that use 30 in the file name to 35.

Note

The banners displayed by the C++Builder installer may include information on features not available in your version of C++Builder.

Project management for packages

See also

A new dialog facilitates editing of packages. With a package selected in the Project Manager, choose Project|Add to Project (or right-click on the package and choose Add). The Add dialog contains four tabs:

- To add a unit to the Contains list, click the Add Unit tab. When you select a .CPP file, C++Builder inserts `USEUNIT("`*`fileName`*`.cpp");` into your .CPP file.
- To add a package to the Requires list, click the Requires tab. When you select a .BPI file, C++Builder inserts `USEPACKAGE("`*`packageName`*`.bpi");` into your .CPP file.
- The New Component tab works like the Component|New Component dialog.
- The Import ActiveX tab works like the Componet|Import ActiveX dialog.

Installing components and packages (update)

See also

Before you install custom or third-party components, copy all the necessary files to the appropriate directories. In general,

- .BPL files belong in the Windows\System or CBuilder3\Bin directory. Wherever you put them, they must be found on the Windows path.
- .BPI and .LIB files belong in the CBuilder3\Lib directory. (The .LIB files are required for static linking.)
- .H files belong in the CBuilder3/Include directory.

If you install a package while no other project is open in the IDE, the new package will be added by default to future projects. To prevent this, uncheck the new package in the Design Packages list, select the Default check box, and click OK. (This updates the DEFAULT.BPR file in the CBuilder3\Bin directory.)

Uninstalling components and packages (update)

See also

You can remove components from the IDE by choosing Component|Install Packages, selecting a package from the Design Packages list, and clicking Remove. Be sure to follow this procedure *before* deleting any of the component files from your hard drive; if you try to uninstall a package that cannot be found on your hard drive, the IDE will generate error messages.

Here are some other things to keep in mind when uninstalling components:

- Projects created while the component was installed may still reference the deleted package in their .BPR files, even if they did not use the component. In this case, edit the projects' .BPR files manually, removing any references to deleted .BPI or .BPL files.
- After uninstalling, you can select the Default check box in the Packages dialog to make sure that future projects don't reference the deleted package.

Upgrading to C++Builder 3.0

Upgrading to C++Builder 3.0 has been divided into the following 3 areas:

- General upgrading issues
  This topic covers upgrading issues that are relevant for previous users of either Borland C++ 5.02 or C++Builder 1.0. It includes backward compatibility issues such as using the new IO streams, and new features such as packages.
- C++Builder 1.0 to C++Builder 3.0 issues
  This topic specifically targets backward compatibility issues with C++Builder 1.0.
- Borland C++ 5.0x-to C++Builder 3.0 issues
  This topic covers a variety of conceptual and compatibility issues specifically targeted to users who are new to the C++Builder environment, and who have not previously programmed with the VCL.

General upgrading issues

The topics concern upgrading from either Borland C++ 5.0X or C++Builder 1.0:

- Changes with OBJ files
- C++ class changes
- C++ Standard Library changes
- IO stream operators
- Sharing modules, code, components, and packages with Delphi
- Linking issues regarding packages and DLLs
- Interoperability Issues with Delphi
- C++ Language support for Object Pascal types and concepts

Changes with OBJ files

C++Builder 3.0 is not object file compatible with C++Builder 1.0 or any previous Borland C++ compiler. Object code and library modules originally compiled using another Borland C++ compiler must be recompiled with C++Builder 3.0 before they can be linked into an C++Builder 3.0 application.

C++ class changes

The typeinfo class is now type_info

The typeinfo class has been changed to type_info. Existing code that uses runtime type information must be recompiled. So, if your code refers to typeinfo, you may need to change it to type_info. For example:

```
class typeinfo;
```

should be now:

```
class type_info;
```

#include <except.h>

Changes to the xmsg class

The xmsg exception class has changed. If you have derived any classes from the xmsg class and if you provided any destructors, you must add a throw specifier in the destructor as follows:

```
class my_exception : public xmsg {
public:
  ~my_exception() throw();   // throw must be added to destructors based on
  xmsg
};
```

Changes to the xalloc class

C++Builder no longer throws "xalloc" exceptions if operator **new** fails. Instead it throws "bad_alloc", in accordance with the C++ standard.

bad_alloc class

An allocation function that fails to allocate storage can invoke the currently installed new_handler   If an allocation function declared with an empty exception-specification, throw(), fails to allocate storage, it returns a null pointer.   All other allocation functions that fail to allocate storage throw a bad_alloc exception (or an exception of a class   descended from std:: bad_alloc).

"string" class

For changes to the “string” class see C++ Standard Library changes.

C++ Standard Library changes

Code changes due to upgrading to the Standard C++ Library, 2.0

C++Builder comes with the Standard C++ Library 2.0 from RogueWave. This C++ library is ANSI-compliant. While most C++ programs are likely to work as they did in the past, you may have to make some changes to C++ programs that use constructs which are no longer part of the ANSI standard.

For instance, the standard library function set_terminate used to take a terminate_function parameter and return a terminate_function.   Now, set_terminate takes and returns a terminate_handler.

For details on the new Standard C++ Library, see the Standard C++ library documentation accompanying the release. Currently, this documentation is available as Word documents in the C++Builder\Help directory. The files include: readme.doc, frontmat.doc, sl-ugv2.doc, sl-cr-1.doc, sl-cr-2.doc, sl-cr-3.doc, sl-cr-4.doc, sl-cr-5.doc, backmat.doc, interna.doc, iosl-cr.doc, iostream.doc.

The following are changes in the C++ Standard Library that are relevant to C++Builder 3.0:

Templates

- Explicit specialization now requires the declarator "template<>" if "-A" is enabled.

I/O Streams

- istream_iterator now takes a second argument, which specifies the   base type of the istream. For example, "istream_iterator<int, char>".

Algorithms

- “accumulate" is now located in <numeric>, instead of <algorith>.
- "times", in <function>, is now called "multiplies".

Exception handling

- The "xmsg" class still exists, but is deprecated (and marked obsolete). Use the "exception" class (or something derived from it) instead.

"string" class

- The new STL "string" class is now accessed by doing **#include** <string>. **#include** <string.h> and **#include** <cstring> both bring in the "C" string functions (like strlen, strcpy, strcmp,...) and **#include** <cstring.h> will still provide the new STL string class (for backwards compatibility).
- The "illegal position" character for the "string" class is now "string::npos", and not NPOS.   For example:

```
if (string().find("mytext") != string::npos) { ... }
```

- "string::remove()" has been renamed to "string::erase()"
- "string::is_null()" no longer exists.   Use "string::empty()" instead.

IO stream operators

Insertion/extraction operators of VCL classes are not visible unless a macro, VCL_IOSTREAM, is defined.

If you are upgrading from C++Builder 1.0, VCL previously included iostream.h. It does not automatically include it now. If you need to include the insertion/extraction operators of the VCL classes, old code that relied on VCL bringing in iostream.h, or that utilized the insertion/extraction operators of VCL classes, must now define VCL_IOSTREAM.

This issue concerns only the C++ classes, declared in the following header files.   These classes were created to support the Delphi built-in language types, such as Currency, Variant, and AnsiString.

- sysdefs.h
- wstring.h
- dstring.h

Because vcl.h includes these header files, this issue also applies to vcl.h.

Interoperability issues with Delphi

C++Builder developers who are using Delphi source code should be aware of the following interoperability issues:

Packages

To install Delphi components create a new package in C++Builder 3.0 and add the .PAS files to the package project. It is necessary to add the following directive to the Pascal source:

```
{$ObjExportAll On}
```

It is also possible to use the DCC32 compiler that comes with C++Builder 3.0 to compile Delphi 3.0 packages. This is a 2 step process as follows:

```
1)dcc32  delphipack.dpk
```

this will result in a BPL which can be installed in the C++Builder IDE

```
2)dcc32 -jphn DelphiPack.DPK
```

this will result in a BPI, OBJ and HPP files for use in C++Builder 3.0

DELPHICLASS and DELPHIRETURN macros

The implementation of the #define macros DELPHICLASS and DELPHIRETURN have changed. As long as your programs used the macros as documented, your code will work as expected. Internally, the C++Builder 1.0 declspec arguments __declspec(delphiclass) and __declspec(delphireturn) are now replaced by __declspec(delphiclass, package) and __declspec(delphireturn, package) respectively.
For more information, see the declspec keyword extension.

Resource strings

For more information about upgrading issues with ResourceStrings, see “Using constants” under Converting C++Builder 1.0 code.

Default parameters

Upgrading from C++Builder 1.0 to 3.0

Topics:

Updating C++Builder 1.0 projects

When you load an old project the makefile is automatically updated and lib files are added to the lib line. Note that these updates apply only to projects and not to specific files. For more information on updating old applications to use packages, see About Packages.

It is recommended to manually add the PACKAGE keyword both as a class modifier in the C++Builder-1.0-generated header and to the “extern” of the form for the following situations:

- If you are going to put your form in a package
- If you choose the IDE option to add a file to a project
- If you both USE and **#include** a C++Builder-1.0-generated form

For projects that used the sample components, see also “Example component name changes” in Updating C++Builder 1.0 components. For information about specific issues updating code from C++Builder 1.0, see Converting C++Builder 1.0 code.

Updating C++Builder 1.0 components

See also

C++Builder 1.0 components require modification and recompilation before they can be installed in C++Builder 3.0. If you don't have access to the source code, contact the vendor who supplied the components.

PACKAGE modifier

The PACKAGE modifier must appear in all declarations of the component class, including forward and friend declarations, and in the declaration of the Register function. PACKAGE is a macro defined in Sysdefs.h that allows classes to be imported and exported from a BPL file. Omitting PACKAGE from class declarations will result in access violations at runtime.

Where you find declarations like

```
class SomeComponent;
friend class SomeOtherComponent;
```

change them to

```
class PACKAGE SomeComponent;
friend class PACKAGE SomeOtherComponent;
```

In the CPP file where the component is defined, include the PACKAGE macro in the declaration of the Register function:

```
void __fastcall PACKAGE Register()
```

Multiple .CPP source modules

C++Builder 3.0 requires a matching header file for each source file where a component is registered. If a component is registered in COMP1.CPP, you need a corresponding header file called COMP1.H.

Some older components—including sample components that shipped with C++Builder 1.0—are registered in a single .CPP file that #includes other .CPP files. When compiled with C++Builder 3.0, such files produce incorrect runtime type information. To fix these components, modify the main registration .CPP file so that it #includes only header files. Then create a new package project in the C++Builder 3.0 IDE that contains each of the .CPP files, making sure that the #pragma package(smart_init) directive appears (after the #includes) in each .CPP file.

Trouble-shooting component installation

If a component is not available in the IDE after installation, here are some problems to check for:

- The PACKAGE modifier is missing in the Register function or in the class declaration.
- The **#pragma package(smart_init)** directive does not appear in a .CPP source file.
- The Register function is not found in a name space with exactly the same name as the source-code module.
- Register is not being successfully exported. Use the TDUMP utility to look for the exported function:

```
tdump -ee=register mypack.bpl
```

TDump displays all exported functions containing the word "register" in them. You should see something like:

```
EXPORT ord:0006='MyComponent::Register() __fastcall'
```

Debugging installed componets

To debug components, you can use the integrated debugger to launch C++Builder:

1. Choose Project|Options, select the Directories/Conditionals tab, and set Debug Source Path to your component source code. The package project does not have to be open in the IDE.

2. Choose Tools|Environment Options and select the Debugger tab. Under Exceptions, select the exceptions you want.

3. Open the component source and set breakpoints.

4. Choose Run|Parameters and set Host Application to CBuilder3\BIN\BCB.EXE.

5. Click the Load button. You should see the CPU window.

1. Choose Run|Run to launch BCB.EXE.

Example component name changes

For C++Builder 1.0 projects that use the example components, note that the classnames for these sample components have been renamed.   For example, TCalendar is now TCCalendar, TDirectoryOutline is now TCDirectoryOutline, and so on.

Updating makefiles

Various changes and additions have been made to the parts of the makefile that are used when building from the command line via **MAKE**. The makefile will still work when it's converted. However, if you are using some of the new features (such as intermediate output directory, new compiler options, and new linker options) the easiest approach is to update the makefile by replacing the static section of the C++Builder 1.0 makefile with the static section from a C++Builder project.

There are comments that have been added to the generated makefile that marks the 'static' (non-IDE-managed) section. The static section of a C++Builder 1.0 project is everything below the .autodepend directive. So you can simply cut and paste from the C++Builder 3.0 section to the C++Builder 1.0 section of the makefile.

This will not affect users who are only building in the IDE. Even from the command-line, the application will build correctly, as long as you do not use the new 3.0 features mentioned above. If you do, the build will either fail (setting an intermediate output path) or will not have the same result as an IDE build (using the new compiler/linker options).

Converting C++Builder 1.0 code

Using Constants

Constants found in CONSTS.HPP have changed to support dynamic load of constants at runtime. The old constants have now been changed to a macro along the lines of UnitName_OldConstantName. That macro actually does a 'LoadResourceString' call which then locates the string in the package that brought that unit in.

Example code changes:

```
throw EInvalidOperation(SInvalidImageSize);
```

becomes

```
throw EInvalidOperation(Consts_SInvalidImageSize);
```

and

```
AnsiString TheText = LoadStr(SFB);
```

becomes

```
AnsiString TheText = Consts_SFB;
```

For more information about ResourceStrings see Resource strings

Nested property assignments

You can no longer use the result of a property assingment as an rvalue. Code like the following will not compile and the assignment must be broken up.

```
__fastcall MyControl::MyControl(TComponent* Owner):TWinControl(Owner)
{
  Height = Width = 10;
}
```

C++ Error: Cannot use the result of a property assignment as an rvalue.

Changes to Pascal types

Changes have been made to some Object Pascal types. These changes can affect users whose code depends upon the signature of a function that uses one of these types. For example, the Object Pascal **Integer** and **Longint** types are now equivalent (**int**), whereas in C++Builder 1.0 they were distinct types translated in the header files as **int** and **long**, respectively. These type changes will not normally be a problem if you do not mix header files from version 1.0 and version 3.0 of C++Builder. Most situations will generate an error.

One occurance of this problem that generates a compiler warning, rather than an error, is overriding virtual functions. If you derived from a class that has a virtual member function that takes a **Longint**, and you override that member function in your class to take a **long**, your member function will not be called virtually. You will get a warning stating that your function hides the virtual function of the base class.

The types that have changed are listed below. You can also compare the typedefs in sysdefs.h between versions.

Type	Size/Values	3.0 C++ implementation	1.0 Implementation
LongInt	32-bit integer	int	long
Cardinal	32-bit unsigned integer	unsigned int	unsigned long
LongBool	true/false or 32-bit unsigned integer	BOOL (WinAPI)	bool
Real	32-bit floating point number	double	float
Comp	64-bit floating point number	Comp class	double

Header file changes

Changes with VCL OLE headers

C++Builder 1.0 projects that use either Oleauto.hpp or Ole2.hpp will require some modification before being rebuilt using C++Builder 3.0.

For automation controllers/servers that include Oleauto.hpp, the easiest upgrade path is to replace the include of Oleauto.hpp with Comobj.hpp:

```
// #include <Oleauto.hpp>
#include <Comobj.hpp>
```

If Ole2.hpp is being used simply to gain access to OLE system functions, this include may be able to be replaced with an include of the standard OLE header Ole2.h. However, note that some code modifications may be required to complete the conversion process.

In order to continue to use either Oleauto.hpp or Ole2.hpp in C++Builder 3.0 projects, the object files for these units must be explicitly added to the project makefile's ALLOBJ macro link list, for example:

```
ALLOBJ = c0w32.obj $(PACKAGES) $(OBJFILES) oleauto.obj ole2.obj
```

These units are no longer part of the standard VCL libraries.

In addition, where the compiler detects identical signatures between functions declared in a version 3.0 standard VCL unit and one of these units, the call will have to be explicitly disambiguated for the project to compile. For example, if including Oleauto.hpp and using the function CreateOleObject(), Oleauto namespacing must be added to the call:

```
// Variant wordObj = CreateOleObject("Word.Basic");
Variant wordObj = Oleauto::CreateOleObject("Word.Basic");
```

- You can no longer include just DSTRING.H. You need to include \VCL\SYSDEFS.H instead of DSTRING.H. Any CPP file that includes only \VCL\DSTRING.H will fail to compile under C++Builder 3.0 This is due to circular inclusion dependencies.

Changes with stdio and stdlib headers

The vcl.h file no longer includes stdio.h and stdlib.h.

Changes with Quick Reports headers

The C++Builder 1.0 header QUICKREP.HPP has been renamed. It is now QUICKRPT.HPP.

Changes with VCL emulation class headers

For information about changes regarding the header files declaring the VCL emulation classes, see IO stream operators.

Upgrading from C++ 5.02 to C++Builder 3.0

The C++Builder compiler can compile most Win32 C and C++ code that is compatible with Borland C++ 5.0.

Note: C++Builder cannot compile 16-bit Windows or DOS programs.

Because of C++Builder's unique exception handling mechanism, object code and library modules originally compiled using Borland C++ 5.01 or earlier must be recompiled with C++Builder's compiler before they can be linked into an C++Builder application. Also, object code modules compiled with the C++Builder compiler do not link into projects built with Borland C++ 5.01 or earlier versions.

C++Builder provides a non-VCL dependent multi-threaded runtime library (RTL) to support legacy applications. This library, called CW32MT.LIB, does not support the VCL enhancements to catching operating system   exceptions since doing so would require the use of the VCL.

The following topics pertain to upgrade issue from Borland C++   to C++Builder:

Converting IDE projects to BPR projects

Visual Database Tools

C++ vs. Object Pascal object models

Visual Database Tools

Visual Database Tools are not supported in C++Builder 3.0. Borland C++ 5.0 Projects that contain these controls/classes, will not convert to C++Builder 3.0 projects.

The VCL classes/controls can be used in place of many of the VDBT classes/vbx's.