RAD Studio XE3

The Developer Force Multiplier

Windows 8

Mac OS X Mountain Lion

C++11

64-bit

Metropolis UI

C99

Boost

Visual LiveBindings

C++

Bjarne Stroustrup

- C with Objects (1979)
  - Modeled OO after Simula and Ada
    - But syntax and RTL based on C
  - Classes
  - Inheritance
  - Inlining
  - Default arguments
  - Type checking
  - CFront compiler

New Turbo C++ Professional

When the object is programming

Be Objective

Object-Oriented Programming (OOP) *is* programming in the '90s. It's the next step after structured programming and is the best way to write applications. So Borland combined the power of OOP with the efficiency of C to produce new Turbo C++ Professional.

And Turbo C++ Professional is the first Turbo-charged native code C++ compiler that brings Object-Oriented Programming to your PC. Since Turbo C++ Professional also compiles ANSI C code, you can be productive with C now, and move to C++ at your own pace.

Environment ++

The best compiler deserves the best environment, and our new Programmer's Platform™ environment makes you more productive. It features overlapping windows and mouse support. And sports a new multi-file editor, an integrated debugger, and a smart project manager. Its advanced open architecture lets you integrate the tools you need to feel right at home.

VROOMM adds room

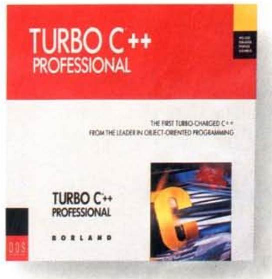
VROOMM™ (Virtual Runtime Object-Oriented Memory Manager) lets you break the 640K barrier. Just select the application code you want to overlay, and VROOMM does the rest—swapping modules on demand. It's fast, easy, automatic.

Another +

Turbo C++ Professional gives you all the tools you need to build fast, reliable C++ programs.

Turbo Debugger® 2.0 debugs your object-oriented programs. This powerful new version is the first and only debugger to support *reverse execution*. Letting you step backwards through your code to find the bugs you might have missed.

New Turbo Profiler,™ the world's first interactive profiler, displays histograms of your program's performance. With it, you can easily spot execution bottlenecks, and see where improvements or redesign of your code will yield maximum performance gains.


TURBO C++
PROFESSIONAL
THE FIRST TURBO-CHARGED C++
FROM THE LEADER IN OBJECT-ORIENTED PROGRAMMING
TURBO C++
PROFESSIONAL
BORLAND
DOS

And Turbo Assembler® 2.0 lets you replace time-critical segments of your code using the world's fastest MASM-compatible assembler.

Turbo C++ Professional Compiler
- C++ conforming to AT&T's 2.0 specification
- C++ class libraries
- Full ANSI C compiler
- VROOMM overlay manager
- Complete documentation and tutorials

Programmer's Platform
- Open architecture for integration of your own tools
- Overlapping windows with mouse support
- Multifile, macro-based editor
- Smart project manager provides visual MAKE
- Integrated debugging and hypertext help

Turbo Debugger 2.0
- Class hierarchy browser and inspectors
- Reverse execution provides "true" undo
- 286 protected-mode and 386 virtual-mode debugging
- Keystroke record and playback

NEW Turbo Profiler
- Displays histograms of program execution
- Tracks call history, overlays, interrupts, file I/O

Turbo Assembler 2.0
- Multipass assembler with NOP squishing and 486 support

Special Introductory Offer

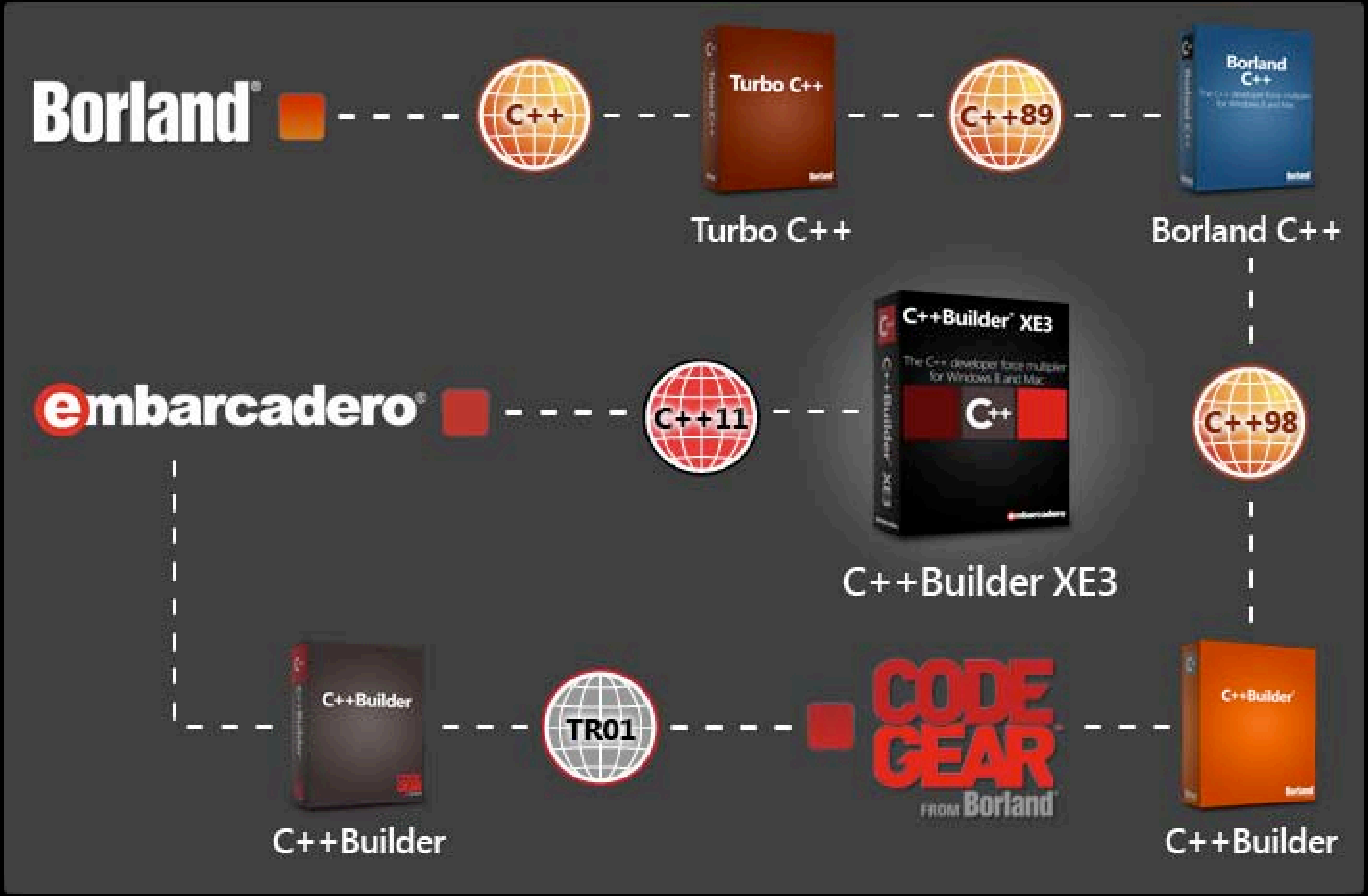
The suggested retail price for Turbo C++ Professional is $299.95 ($199.95 for Turbo C++). For a limited time, Borland is offering its dealers and distributors special introductory discounts.* So be objective, and **SEE YOUR DEALER** or call Borland** at **1-800-331-0877** now!

BORLAND

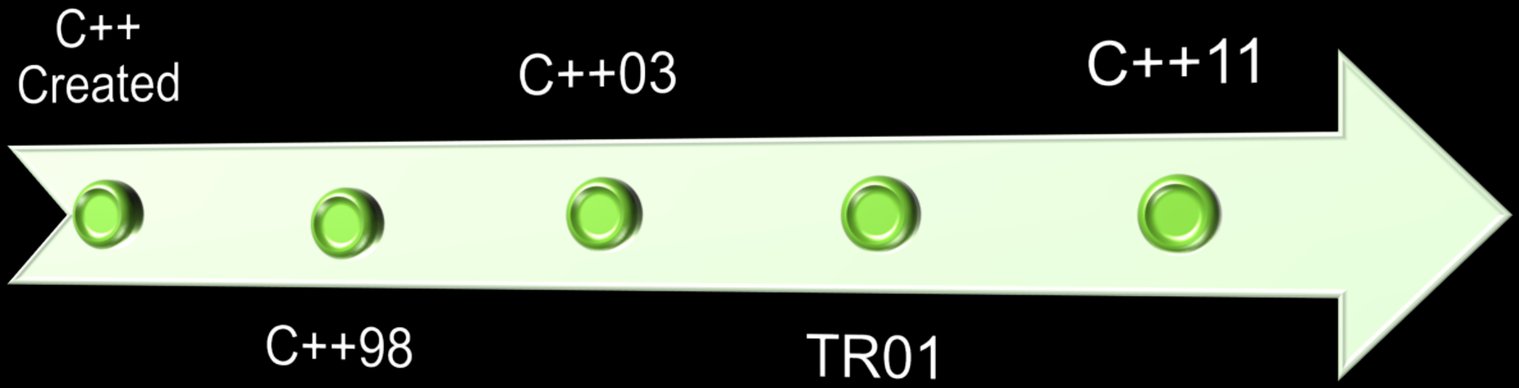
Code: MC66

*Offer expires July 31, 1990 or while supplies last. Offer good in United States and Canada only. **Special discounts for registered Turbo C® owners are available from Borland. Mail orders to: Borland, P.O. Box 660001, Scotts Valley, CA 95067-0001. For orders outside the U.S., call (408) 438-5300. Turbo C++, Turbo Debugger, Turbo Profiler and Turbo Assembler are trademarks or registered trademarks of Borland International, Inc. Copyright © 1990, Borland International, Inc. All rights reserved. BI-1333

Path to C++


Borland
C++
Turbo C++
Turbo C++
C++89
Borland C++
Borland C++
embarcadero
C++11
C++Builder XE3
The C++ developer force multiplier for Windows 8 and Mac
C++Builder XE3
C++98
C++Builder
C++Builder
TR01
CODE GEAR FROM Borland
C++Builder
C++Builder

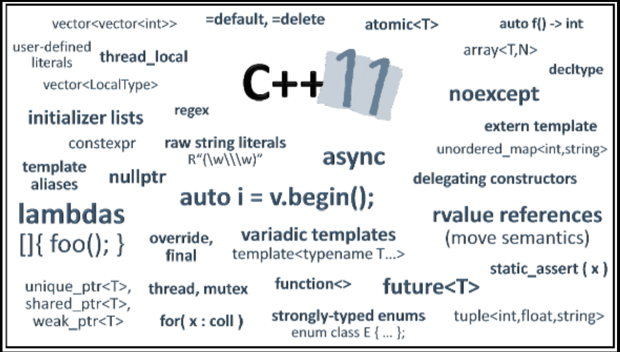
A Brief History of C++



C++11 – A new Standard

Language

- Rvalue references and move constructors
- constexpr - Generalized constant expressions
- Core language usability enhancements
- Initializer lists
- Uniform initialization
- Type inference
- Range-based for-loop
- Lambda functions and expressions
- Alternative function syntax
- Object construction improvement
- Explicit overrides and final
- Null pointer constant
- Strongly typed enumerations
- Right angle bracket
- Explicit conversion operators
- Alias templates
- Unrestricted unions



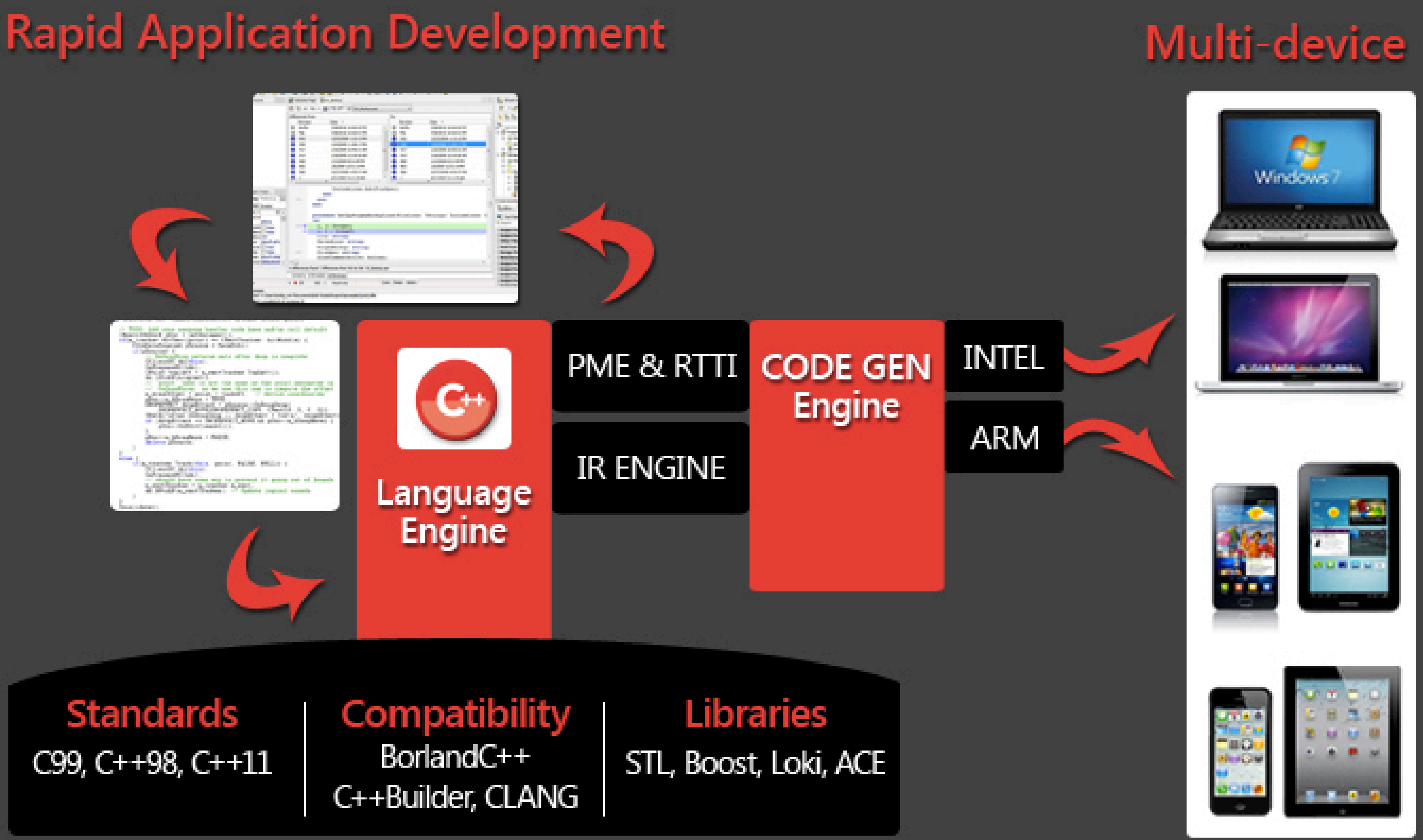
Library

- Variadic templates
- New string literals
- User-defined literals
- Multithreading memory model
- Thread-local storage
- Explicitly defaulted and deleted special member functions
- Type long long int
- Static assertions
- Allow sizeof to work on members of classes without an explicit object
- Control and query object alignment
- Allow garbage collected implementations
- Threading facilities
- Tuple types
- Hash tables
- Regular expressions
- General-purpose smart pointers
- Extensible random number facility
- Wrapper reference
- Polymorphic wrappers for function objects
- Type traits for metaprogramming

64-bit C++Builder for Windows

- C++11 support in BCC64 compiler
- VCL and FireMonkey
- Dinkumware STL for C++11/C99 version 5.30
- Boost version 1.50.0
- Highly-optimized code generation

An Architecture for the Future



Standards
C99, C++98, C++11

Compatibility
BorlandC++
C++Builder, CLANG

Libraries
STL, Boost, Loki, ACE

VCL and FireMonkey

VCL

- 32-bit and 64-bit Windows applications
- Windows 8 Metropolis UI
- Non-client area styling
- Sensor API

FireMonkey

- 32-bit and 64-bit Windows applications
- Windows 8 Metropolis UI
- Mountain Lion and Retina
- Mac OS X app store compatible
- Sensor API and non-client area styling

C++11

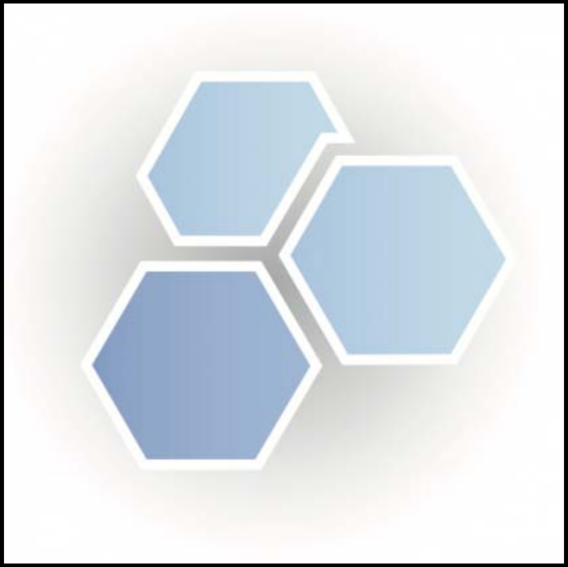
- Language
  - auto
  - ranged-for loop
  - lambda expressions
  - uniform initialization syntax
  - variadic templates
  - rvalue references
  - delegating constructor
  - thread local storage
  - in-class member initialization
  - And much more!
- Library
  - random number generators
  - new auto and shared pointers
  - hash map
  - atomic operations
  - regular expressions
  - async
  - threads
  - metaprogramming and traits
  - And much more!

Dinkumware Standard C++ Library

- Includes:
  - Standard Template Library (STL)
  - Standard C Library
  - Standard C Library Headers
- Version 5.30 – C++ 64
- Version 5.01 – C++ 32
- http://www.dinkumware.com/

Boost Libraries

- Some new C++ features begin life in Boost
- As usage grows, adopted into C++ language or STL
  - boost::bind -> lambda expression binding
  - boost::for_each -> ranged for loop
- Two versions
  - 1.50.0 – 64-bit C++
  - 1.39 – 32-bit C++



BCC32 and BCC64

- size_t versus unsigned
  - size_t is defined as an unsigned integral type
  - In Win32 and Win64, this is the same size as a pointer
  - In Win32's "ILP32" data model, int (and long) and pointers are 32-bit. You could use unsigned int in place of size_t, although it was not portable.
  - Win64 is an "LLP64" data model: long long and pointers are 64-bit, while int (and long) are still 32-bit. Therefore, you must use size_t.
- _WIN32 Is Defined For Win64
  - _WIN32 is defined (as the integer 1) for both Win32 and Win64 targets. This allows programs to target (modern) Windows in general, among other platforms.
  - _WIN64 is defined only for Win64 targets

BCC32 & BCC64 – Windows Programming

- 64-bit Windows Applications use the familiar Windows API
- Windows API calls must be 64-bit versions.
- Try blocks are supported in 64-bit Windows programs.
- A 64-bit Windows application can use a 32-bit Windows type library (as some 64-bit MS Office applications do).
  - Cannot mix 32-bit and 64-bit code in the same process.
- DLLs, components, libraries, and packages require that you compile or install separate 32-bit Windows (design-time) and 64-bit Windows (run-time) versions if you want to use the Form Designer.
- 64-bit Windows is needed for OS extensions, shell extensions.
- The size of LRESULT, WPARAM, and LPARAM all expand to 64 bits, so message handlers will have to be checked for inappropriate casts.

BCC64 - Compiler

- BCC64 is based on the Clang compiler front-end.
  - Different set of compiler options
  - BCC64 is more compliant with C++ language standards than BCC32.
- In addition to new, more specific and detailed warnings and error messages, BCC64 phrases messages for conditions detected by BCC32 in a different way.
- To get all the predefined macros directly from the preprocessor, run: echo | bcc64 -E -dM –
- Detecting BCC64: check for _WIN64. To detect BCC64 specifically, you can use:
  - #if __BORLANDC__ && __clang__
  - __BORLANDC__ is the compiler version (currently 0x0650 for version 6.50)
  - __clang__ is 1 for BCC64.
- #include Paths and Lookup - BCC64 supports three different header/source paths:
  - -isystem is for system headers included with BCC64.
  - -I is for headers provided by third parties.
  - -iquote is for your own source files and headers, #include "file". If the named file is not found, then the paths specified by -I and -isystem are searched, as if the directive was #include <file>

BCC64 - Compiler

- Precompiled Headers work differently
  - Each 64-bit Windows C++ project can have only one precompiled header,
  - A default precompiled header (named projectPCHn.h) is generated for each new C++ project (for any platform)
- Object and Library File Format
  - BCC32 and its associated tools use OMF in .obj and .lib files
  - BCC64 uses ELF in .o and .a files
  - When you migrate a 32-bit Windows application to 64-bit Windows, you must change references to .lib and .obj to be .a and .o, respectively.
- Unicode Identifiers - Although Unicode is supported in literal strings and file names, Unicode in identifiers is not allowed

BCC64 – Assembly Language Programming

- Inline Assembly
  - BCC32-style inline assembly is not supported
  - Functions written entirely in assembly (with a separate assembler) may be linked into your program
  - Clang does support inline assembly, but with line-by-line AT&T syntax, not the more familiar block-of-Intel syntax.
- Most of the registers of a 64-bit Windows CPU are twice as wide as those in a 32-bit Windows CPU. Yet the size of the instruction register (IR) is the same for 32-bit Windows and 64-bit Windows processors.

Upgrading Existing BCC32 Projects

- Object and Library File Format
  - BCC32 and its associated tools use OMF in .obj and .lib files.
  - BCC64 uses ELF in .o and .a files.
  - Where possible, object and library file extensions should be removed. When necessary, as in custom scripts, the extension must be changed or made conditional with version detection.
- #pragma link
  - If the files named in #pragma link statements contain a file extension, those extensions must be removed. Each compiler will append the appropriate extension.
  - For example, Control Panels apps that use this statement:
    - #pragma link "Ctlpanel.obj"
    - must be updated to read:
    - #pragma link "Ctlpanel"
- Applications that use the Windows API must explicitly contain: #include <windows.h>
  - With BCC32, including windows.h is not required, but BCC64 requires windows.h and is more strict about #includes.
- NO_STRICT Macro - The NO_STRICT type checking scheme is not supported in BCC64. If you have existing projects that use it, it should be removed.
- Updating WebBroker Projects
  - Change #pragma link as described above.
- Updating WebSnap Projects
  - Change #pragma link as described above.
  - Change _fastcall (single-underscore) to __fastcall (double-underscore).

64-bit C++Builder for Windows

- C++11 support in BCC64 compiler
- VCL and FireMonkey
- Dinkumware STL for C++11/C99 version 5.30
- Boost version 1.50.0
- Highly-optimized code generation

General Availability

- Download on December 10th
- Re-install for XE3 developers
- Binary compatible with XE3