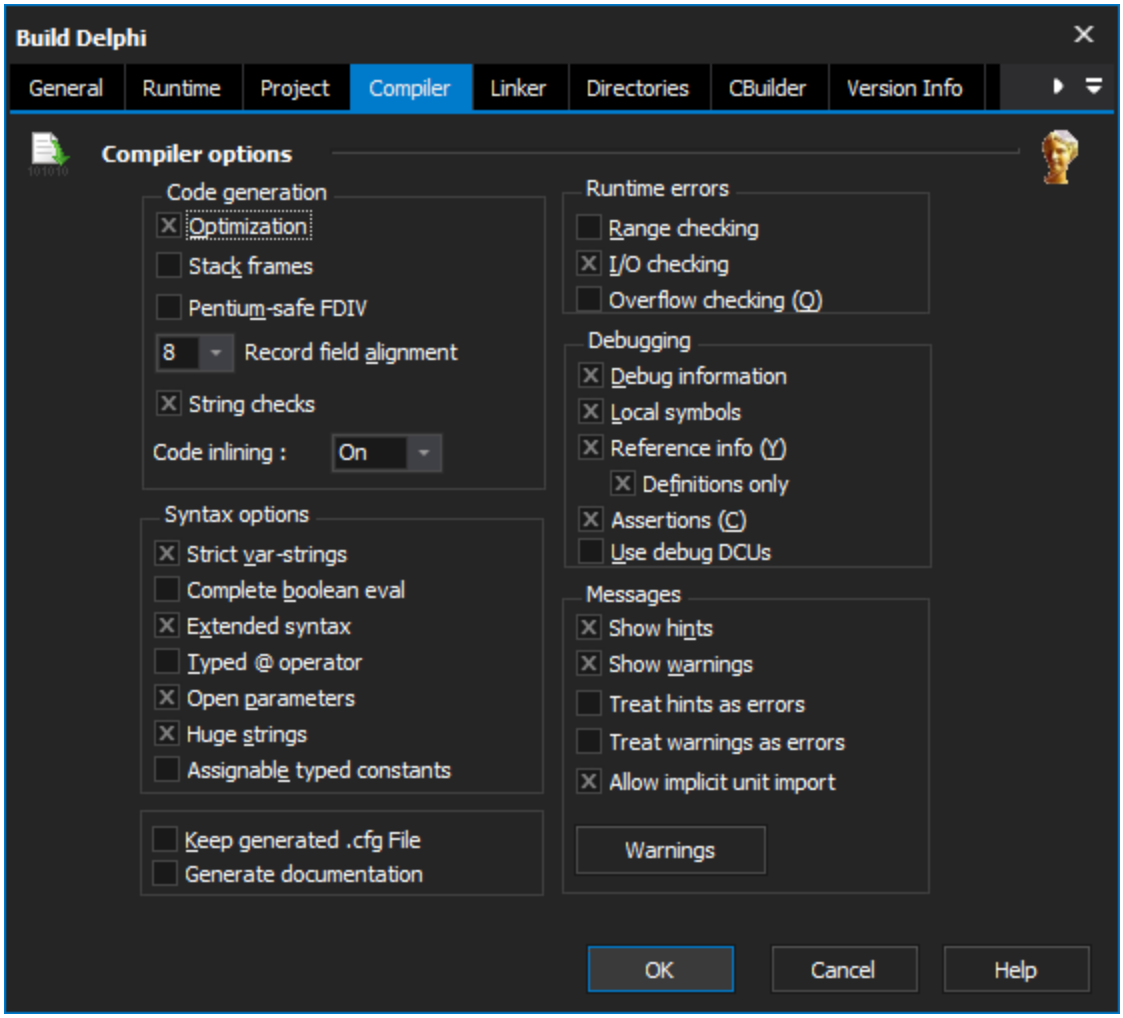
Build Delphi Project Action

This action allows you to automate the compilation of Delphi projects. FinalBuilder supports the following versions of Embarcadero /Codegear Delphi:

Set the compiler version you need from the "Compiler Version" section of the "Project" tab (see below.)

Many of the property pages for this action use properties which are identical to the properties in Delphi IDE. The properties on the Compiler, Linker, Directories and Version Info pages are described in the Delphi Help File. You may use FinalBuilder Variables in the Directories page but not on the Version Info page.

This action can also be used to maintain the version info for the selected Delphi project. When the AutoIncrement property (on the version info tab) is enabled, the version number properties (Major,Minor, Release and Build) are persisted to a file (yourproject.fbd) after the action runs (and after the AfterAction script event). These values will be restored when the project is loaded. If the action fails, the values are not persisted. This makes the AutoIncrement property function the same as it does in the Delphi IDE.



On this page:

Project

The options that are specific to FinalBuilder are provided on the Project property page, pictured above.

Project File

This is the path to the Delphi project, Delphi package or unit file you wish to compile. You must choose the .dpr or .dpk file for a project or package, not the .bdsproj/.dproj/.dof file (all of the latter are project settings files.)

From Project File...

Load Project Settings

The Delphi IDE maintains a file with the extension .dof, .bdsproj or .dproj with the project settings. Press this button to automatically populate all of the action's property pages with the settings from this file. You should do this when you first create the action. Once the settings are loaded, they will not be reloaded into FinalBuilder from the Delphi IDE unless you press this button again, or check the "Use Settings From Project File" option (see below.)

Regenerate Manifest File

Check this option to have the manifest regenerated during the compile.

Icon

Icon file

This is the path to the icon that will be used as the main icon for the application. When you select a project file, FinalBuilder checks to see if there is a resource file with the same name as the project (ie the default one generated by the Delphi IDE). If the file exists, FinalBuilder extracts the MAINICON resource and saves it in the same folder as mainicon.ico. This is needed for the resource compiler.

Compiler Version

Compiler Version

Choose the version of Delphi to use to compile.

Work around Delphi 5 Compiler Bug

Check this property (which is only enabled when the compiler version is D5) to work around the problem where the project compiles fine but no executable is produced. The will save having to work around it manually by adding a second action.

Use EurekaLog Compiler

Check this property to use the EurekaLog compiler instead of Delphi. See the Using EurekaLog topic for more information.

Platform

Choose the platform to compile for. This selects the correct global library paths for the selected platform, and adds the platform output path. Only available for compiler greater than XE2.

Options Source

Load Settings from project file

Checking this box will cause FinalBuilder to ignore any settings made on the action's property pages, and instead always use the values from the project settings file. You can choose to apply this option only to certain groups of settings, any combination of Packages, Compiler, Linker, Directories and/or Version Info can be loaded from settings. Checking this option disables the relevant property page(s).

Update Project Settings File with Version Info

Check this option to have the project file update with the version information used in the compile.

Project Config

Select which configuration of the project you want to build. (Delphi 2007 only)

Options

Build All

Tells the command line compiler to rebuild all files in the project. This is checked by default. When not checked, the command line compiler will only recompile modified units.

Regenerate Resource

When checked, FinalBuilder will regenerate the project resource file before compiling the project. It calls the Embarcadero Resource Compiler to do this. You should leave this option checked if you use version info in your project and change any version numbers between builds

Update Package Source

Enable this option to update the package source with changes when the action runs.

Build with runtime Packages

This is the same as the equivalent option in the Delphi project options.

Starting Dir

Select the starting directory for the action.

Extra Command Line Options

Enter any extra command line options to be passed to dcc32.exe.

You need write access to the directory you are compiling the project from, as FinalBuilder generates temporary .cfg file for the command line compiler to use.

Compiler

Code Generation

Optimizations

When turned on the Delphi compiler will perform code optimisations. Examples of these are (but not limited to):

- Placing variables in CPU registers
- Eliminating common subexpressions
- Generating induction variables

All optimization operations are considered "safe" and don't alter the meaning of the program.

Stack frames

When turned on stack frames are created for every procedure and function in the source code for the project. When turned off stack frames are only generated when they are required.

Pentium-safe FDIV

When turned on the compiler will generate floating point code which will protect again the FDIV flaw in Pentium processors. Some operating systems correct this issue system wide and therefore don't require this fix (Win95, WinNT, and higher).

Record field alignment

Determines the maximum padding that can be used to align a field. Each setting has some extra side affects these are listed below;

- 1: Fields are not aligned, all structures are packed.
- 2: Fields in non-packed records and in class structures are aligned on word boundaries.
- 4: Fields in non-packed records and in class structures are aligned on doubleword boundaries.
- 8: Fields in non-packed records and in class structures are aligned on quadword boundaries.

Variables and typed constants are all aligned with optimal access in mind.

String Checks

Determines whether the compiler generates code which checks the assignment of strings. This is only used by Delphi 2010 and earlier compilers.

Code Inlining

Determines whether code will be placed inline or left as an external call. This option is only a suggestion to the compiler as some situations mean that code can not be made inline. The possible values for this options are;

On: The routine will be compiled as inline if the inline directive is used.

Off: The routine will not be compiled as inline, even if the inline directive is used.

Auto: The same as On. Added to this is that if the routine is 32 bytes or less in size it will also be compiled as inline.

Syntax options

Strict var-strings

When turned on the compiler will enforce type checking for short strings passed as variable parameters. This is really only important for Delphi code which uses short strings.

Complete boolean eval

When turned on the compiler generates code in which all components of an AND and OR statement are evaluated. Therefore guarenteeing calling of each component on the statement.

When off the compiler generates code which short circuits the statements. This means when evaluated in left to right order, if the result of the statement has become apparent no more of the statement is called.

Extended syntax

Determines whether Delphi Extended Syntax should be used or not. When enabled the following Delphi features are able to be used;

- Function Statements
- The Result Variable
- Strings Compatible with PChar

Typed @ operator

Determines how the compiler should treat the @ operator and compatibility of pointer types.

- Enabled: Only pointers of the same type are compatible. All @ operations create a variable which has pointer type specific to the variable the @ was applied to.
- Disabled: All @ operators generate an un-typed pointer. This pointer is compatible with all other pointer types.

Open parameters

Controls the compilers interpretation of variable parameter strings. When disabled variable string parameters are not treated differently. When enabled variable string parameters are treated as open strings. Regardless of this setting OpenString identifier can be used at any-time.

Huge strings

Determines how the compiler interprets string types. When On this options will mean the compiler will use AnsiString, when off the compiler will use ShortString.

Assignable type contants

Controls as to whether the compiler will allow typed contants to be altered or not. When turned On constants are able to be assigned to. When Off constants are not able to be assigned to. In some older code this option is required for the code to compile, for newer projects its suggested to not enable this option.

Keep generated .cfg file

Whether to keep the config generated for the compiler or not. When On this option means the action will keep the cfg file and rename it to <ProjectFile>. used or dcc32.used. The file will be placed in the same directory as the project file being compiled.

Generate documentation

Enables or disabled the generation of documentation generation from XML documentation tags with in the source code.

Runtime errors

Range checking

When enabled all array and string indices are validated to be within the bounds of the variable. In addition all scalar and subrange variables are validated to within their bounds. Having this option enabled adds extra checking to the generated code, and therefore will execute slower.

I/O checking

When enabled all I/O procedures have code generated which checks for non-zero return results. When a non-zero return result is detected an EInOutError exception is raised.

Overflow checking

When enabled the arithmetic operations of +, -, *, Abs, Sqr, Succ, Pred, Inc, and Dec are have results checked for overflowing the returned variables type. Having this option enabled adds extra checking to the generated code, and therefore will execute slower.

Debugging

Debug Information

When enabled the compiler is signalled to generate debugging information which is maps object-code to address into the source text file. This information is stored in the unit file with unit object code. This does not affect the speed of generated code, however does cause the compilation process to take up more memory.

Local symbols

When enabled the compiler generates local symbol information. Local symbol information is the name and type of all local variables and constants in the a module. This information is stored in the unit file with unit object code. This does not affect the speed of generated code, however does cause the compilation process to take up more memory.

Reference info

When enabled the compiler generates symbol reference information. This information is typically used by the Code Editor and the Project manager. Some other tools may require it.

Definitions only

In combination with the Reference Info option, this option simply limits the reference information generated to just when identifiers are defined. If turned off, information would be generated for when identifiers are defined and when they are referenced.

Assertions

Determines whether the compiler should include assert generation or not. When turned On assert calls will be included into the generated code. When turned Off assert calls will be omitted from the generated code.

Use Debug DCUs

When turned on the compiler will switch to using the debug DCU path specified on the Directories tab.

Messages

Show hints

Whether to report hints into the build log or not.

Show warnings

Whether to report warnings into the build log or not.

Treat hints as errors

Set if hints should be treated as an error and cause the compilation to fail if found.

Treat warnings as errors

Set if warnings should be treat as an error and cause the compilation to fail if found.

Allow implicit unit import

If the compiler reports "implicitly imported into package" and this option is turned off then an error will be reported.

Linker

Linker Options

Map File

Linker can generate a map file which contains general segment information about the generated code. The options available are:

- Off: No map file is produced.
- Segments: Generates a map file which includes general segments such as start address. Also includes hints and warnings messages produced during linking.
- Publics: Generates the same information as the segments level with the addition of information about public symbols.

- Detailed: Generates the same information as the publics level with the addition of a resource string file. Added to this the detailed segment map includes the segment address, length in bytes, segment name, group, and module information.

EXE and DLL options

Generate console application

Tells the linker to to flag the applications .exe as a console mode application.

Include TD32 debug info

Tells the linker to generate a line-number table for each procedure for mapping source code to object-code.

Place in separate TDS file

Instructs the linker to place C++ style debug information into a separate TDS file with the same name as the project.

Include remote debug symbols

Instructs the linker to generate remote debug symbol information for generated code. These are .rsm files.

Description

EXE Description

The description string for the generated executable. Only usable for executable and DLL generation.

Memory sizes

Min stack size

Instructs the linker as to the initial committed size of the stack.

Max stack size

Instructs the linker as to the maximum reserved size of the stack.

Image base

Instructs the linker as to the default load address for an application, DLL, or package.

Directories

Directories

Output directory

The directory to which generated binaries will be written. Whether this be a dll, dcu, or exe.

Unit output directory

The directory to which all dcu files will be written. If left blank dcu files will be generated in the same location as the unit they relate to.

Always use Search Path from Project Settings File

If enabled the compiler will be given the search path from the project settings of the supplied project. The search path supplied in this options dialog will be ignored.

Search path

The search path to give to the compiler. The search path is where the compiler will look for source files.

Use Global Delphi Library Path

If enabled the compiler will be given the global library path instead of the library path specified in the action.

Library Path

The paths in which the compiler will look for libraries.

BPL output directory

The directory to which generated package files are written to.

DCP output directory

The directory to which all dcp files will be written. If left blank then the global DCP output directory will be used.

Conditionals

Use Conditionals from Project Settings File

If enabled the conditional defines listed in the project file will be used instead of the ones listed in the dialog.

Conditional defines

The conditional defines which should be passed to the compiler. Each one should be separated with a semi-colon.

Aliases

Unit aliases

A list of name/value pairs that define unit name aliases. Useful for backwards compatibility and where units have either change name, or been merged into one larger unit.

Namespace Prefixes

Specify namespaces to automatically apply to uses clauses for unit names which belong to that scope. E.g. Specify FMX, so that FMX.Graphics can be specified as Graphics.

CBuilder

C++ Builder Output

Link output

Instructs the compiler what it is meant to generate as output.

Generate DCUs

Generates the package and associated .dcu and .dcp files.

Generate C objects files

Generates C object files for linking with C programs.

Generate C++ object files

Generates C++ object files for linking with C++ programs.

Include namespaces

Includes the namespace information into the generated files.

Export all symbols

Includes all symbol information into the generated files.

Generate .hpp files

Adds the generation of .hpp files.

Generate All C++ Builder files

Generates all header files, package import files, and package static library files.

BPI output directory

The directory that .bpi files are written.

Obj/Lib output directory

The directory that .obj and .lib files are written.

HPP output directory

The directory that .hpp files are written.

Version Info

Include version information in project

Whether the linker should include version information into the generated binary or not.

Use Property Set for Version Info

Whether to use a Property Set for the version information of the generated binary or not.

Module version number

Major version

The major version number of the generated binary. Can be set directly, or taken from a supplied Property Set.

Minor version

The minor version number of the generated binary. Can be set directly, or taken from a supplied Property Set.

Release

The release version number of the generated binary. Can be set directly, or taken from a supplied Property Set.

Build

The build version number of the generated binary. Can be set directly, or taken from a supplied Property Set.

Auto-increment build number

Whether to automatically increase the build number portion of the Module Version Number or not. If the number is increased the value is updated into the action. The project will need to be saved so that this persists for the next execution.

Use Version Number from Project Settings File

Whether to use the version number specified by the Project Settings File or not. This is useful if the project file is to keep in lock step with releases from your build machine. Update the version numbers in the project settings and then use these to release the build.

Module Attributes

A binary can be marked with a number of different attributes. These are namely;

Debug build

Indicates that the project was compiled in debug mode.

Special build

Indicates that the version is a variation of the standard release.

DLL

Indicates that the project includes a dynamic-link library.

Pre-release

Indicates the version is not the commercially released product.

Private build

Indicates that the version was not built using standard release procedures

Language

The primary language of the binary.

Code Page

Determines the default text character set that is used in ANSI source code. This is used by the compiler in working out how to parse multibyte character strings. This typically includes;

- String constants
- Comments
- #error directive
- #define directive

Locale ID

The locale ID which relates to the code page selected.

Version Information

All the version information which will be attached to the binary when its generated. These can either be sourced from the Project Settings, Property Set, or set directly in the dialog.

Include Compile Date in Version Info

Where to update the version information to have the date of the action run included into the version information as the compile date.

Auto-Update FileVersion String

Whether to automatically update the FileVersion string based on the FileVersion major, minor, release, and build numbers.

Update Project Settings File with version info

Whether to update the project settings file with the version information used in the compile. Project Settings will have to be checked back into version control if the project is under source control.

Link ProductVersion to FileVersion

Whether to link the ProductVersion file in the version information to the FileVersion field. Typically used if the product has only one version number across all parts of the build.

Auto-Update ProductVersion String

Whether to automatically update the ProductVersion string based on the ProductVersion major, minor, release, and build numbers.

Update VersionInfo Keys setting

Writes the VersionInfo string to the Version Info Keys key in the Project Settings file.

Resource Compiler

Resource Compiler

Compiler

The executable to use for compiling resources.

Resource Compiler Location

When the RC.EXE and Other options are chosen for the resource compiler their location on disk is required.

Resource Compiler Parameters

The way in which parameters should be passed to the resource compiler. The %1% variable will be replaced at runtime with the resource script file name.

Compile Project Resources (D2009 or later)

Whether to include project resources in the resource compile or not.

Manifest

Include Manifest File In Resources

Whether to include the manifest file in the compilation of the resources. If this option is selected the location of the manifest file to include needs to be supplied.

Custom VCL Styles

The location of the VCL styles to include in the resource compile. Leave blank if there are none to include.