Code Insight References

Go Up to The Code Editor Index (IDE Tutorial)

Code Insight refers to a subset of features embedded in the Code Editor (such as Code Completion, Code Parameter Hints, Code Hints, Index for C++ Insights, Block Completion, Help Insight, Class Completion, Error Insight, and Code Browsing) that aid in the code writing process. These features help identify common statements you want to insert into your code, and assist you in the selection of properties and methods. Some of these features are described in more detail in the following subsections.

To enable and configure Code Insight features, choose Tools > Options > User Interface > Editor Options > Source and click on Code Insight option.

> *Note:*
>
> *The **Code Insight** features are now defined per Language.*
>
> *When using Delphi you can use Classic completion.*
>
> *For C++ you can use Classic completion for the Classic compiler only.*

Contents

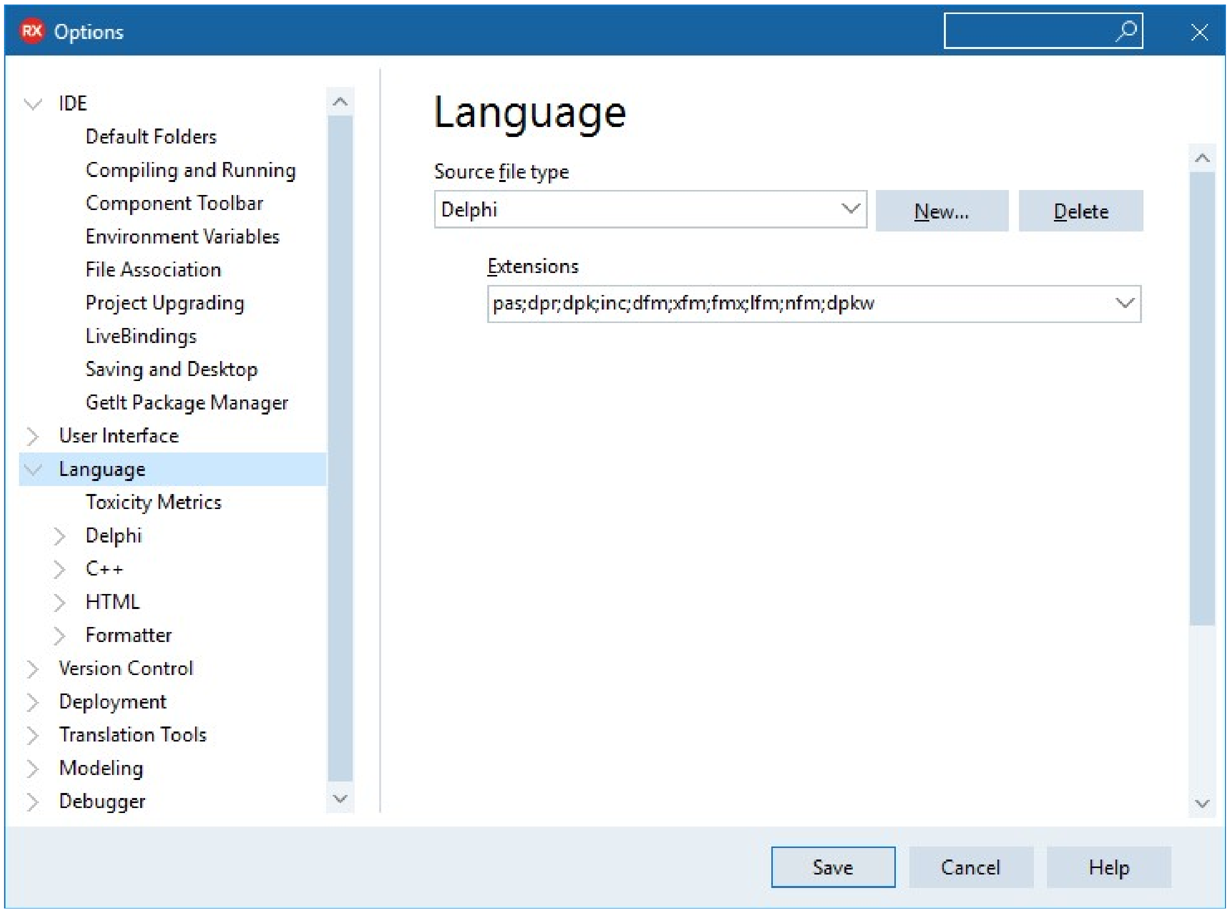
Code Insight using Delphi Language Server Protocol (LSP)

The Language Server Protocol (LSP) is a specification that provides language services through a separate server process that communicates with an IDE. As a result, Code Insight features are provided by a separate process(es), and the features are asynchronously provided.

> *Note: The **Code Insight** now works while debugging.*

Languages

Languages are defined in Options > Language setting pane, where you can see a list of all languages available. You can also create a new language.



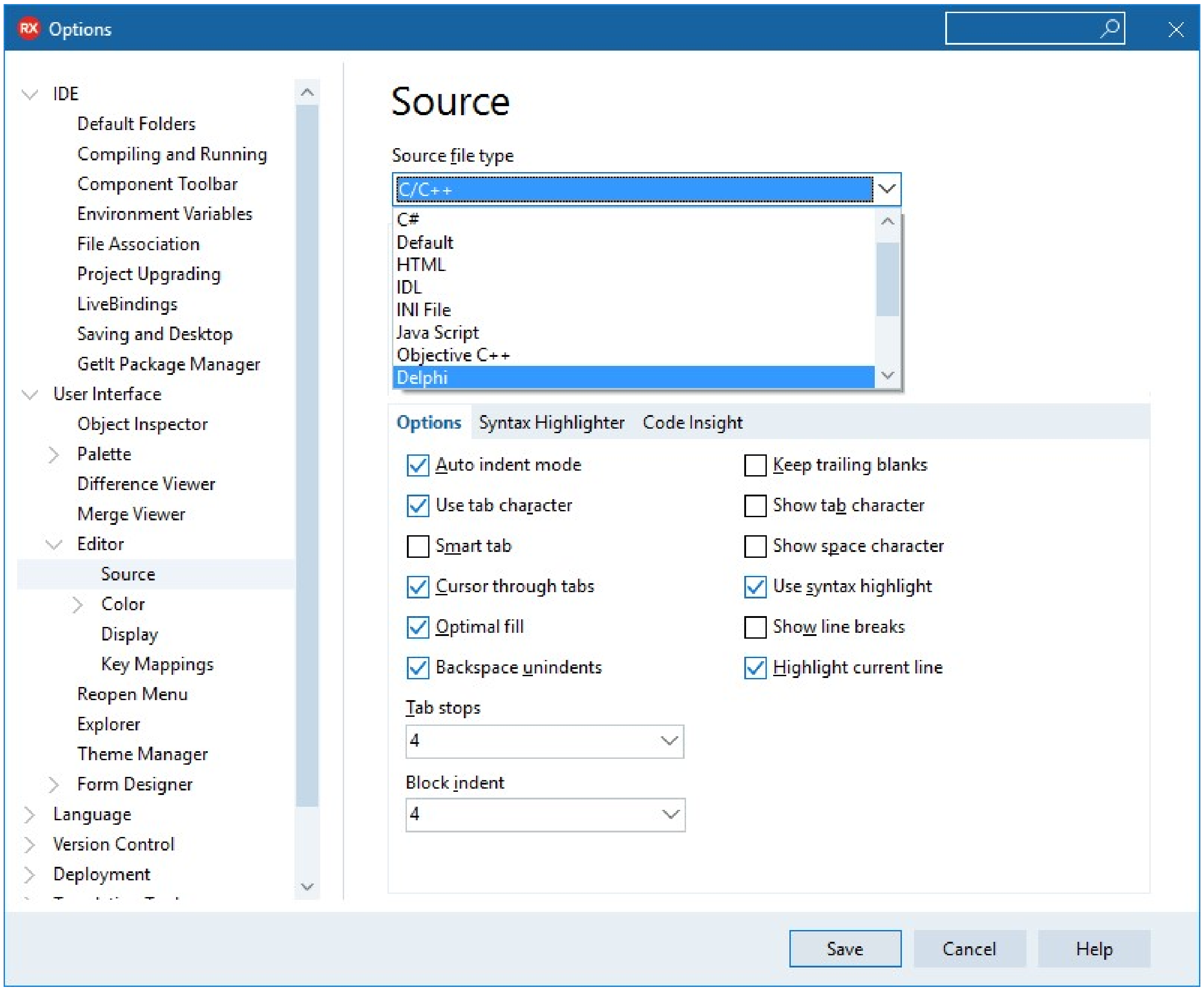
Language

Source file type: Delphi

Extensions: pas;dpr;dpk;inc;dfm;xfm;fmx;lfm;nfm;dpkw

Editor Options and Syntax Highlighting

The Editor options and Syntax Highlighting are defined per language. You can find them in Options > User Interface > Editor options > Source and click on **Options**.


Options

IDE
Default Folders
Compiling and Running
Component Toolbar
Environment Variables
File Association
Project Upgrading
LiveBindings
Saving and Desktop
GetIt Package Manager
User Interface
Object Inspector
Palette
Difference Viewer
Merge Viewer
Editor
Source
Color
Display
Key Mappings
Reopen Menu
Explorer
Theme Manager
Form Designer
Language
Version Control
Deployment

Source

Source file type: C/C++

C#
Default
HTML
IDL
INI File
Java Script
Objective C++
Delphi

Options | Syntax Highlighter | Code Insight

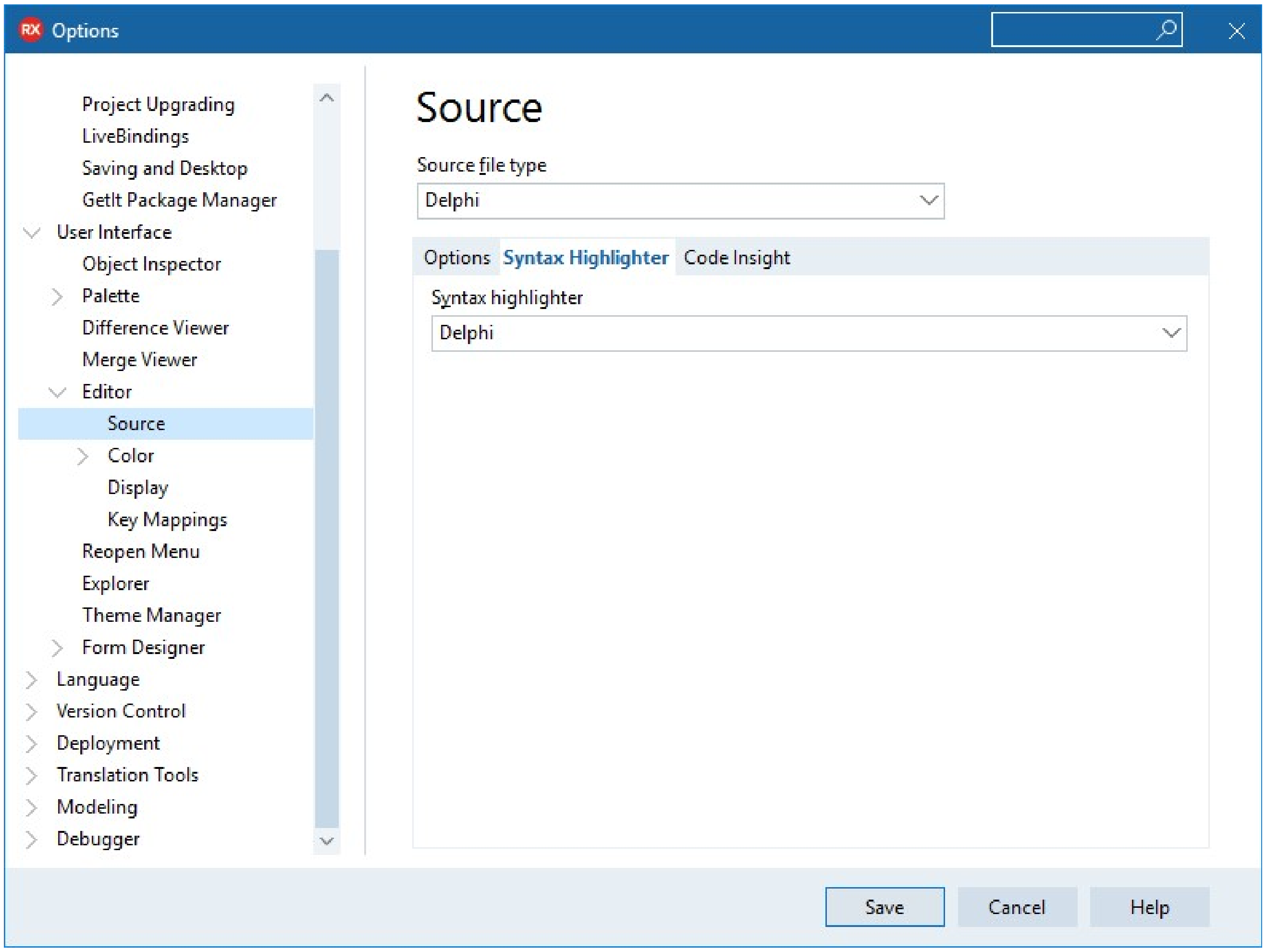
- [✓] Auto indent mode
- [✓] Use tab character
- [ ] Smart tab
- [✓] Cursor through tabs
- [✓] Optimal fill
- [✓] Backspace unindents
- [ ] Keep trailing blanks
- [ ] Show tab character
- [ ] Show space character
- [✓] Use syntax highlight
- [ ] Show line breaks
- [✓] Highlight current line

Tab stops: 4

Block indent: 4

Save | Cancel | Help

On this setting pane, you can choose a language. Then, for that language, there are the editor options, as well as the **Syntax Highlighter** to use for that language.



Source file type: Delphi

Syntax highlighter: Delphi

Changing Code Insight / LSP Manager

The Delphi LSP is enabled by default. You can check if the IDE is using Delphi LSP by opening a Delphi project > Task Manager and look for a **DelphiLSP.exe** process.

The final tab for a language is **Code Insight**. It provides the **User Editor Font** and a set of **Code Insight** settings.

The **Code Insight Manager** provides Code Insight functionality for a language. You can set this to any manager listed there, even if that manager was not intended for the language.

Note:

*The **Code Insight Manager** lists all managers registered with the IDE.*

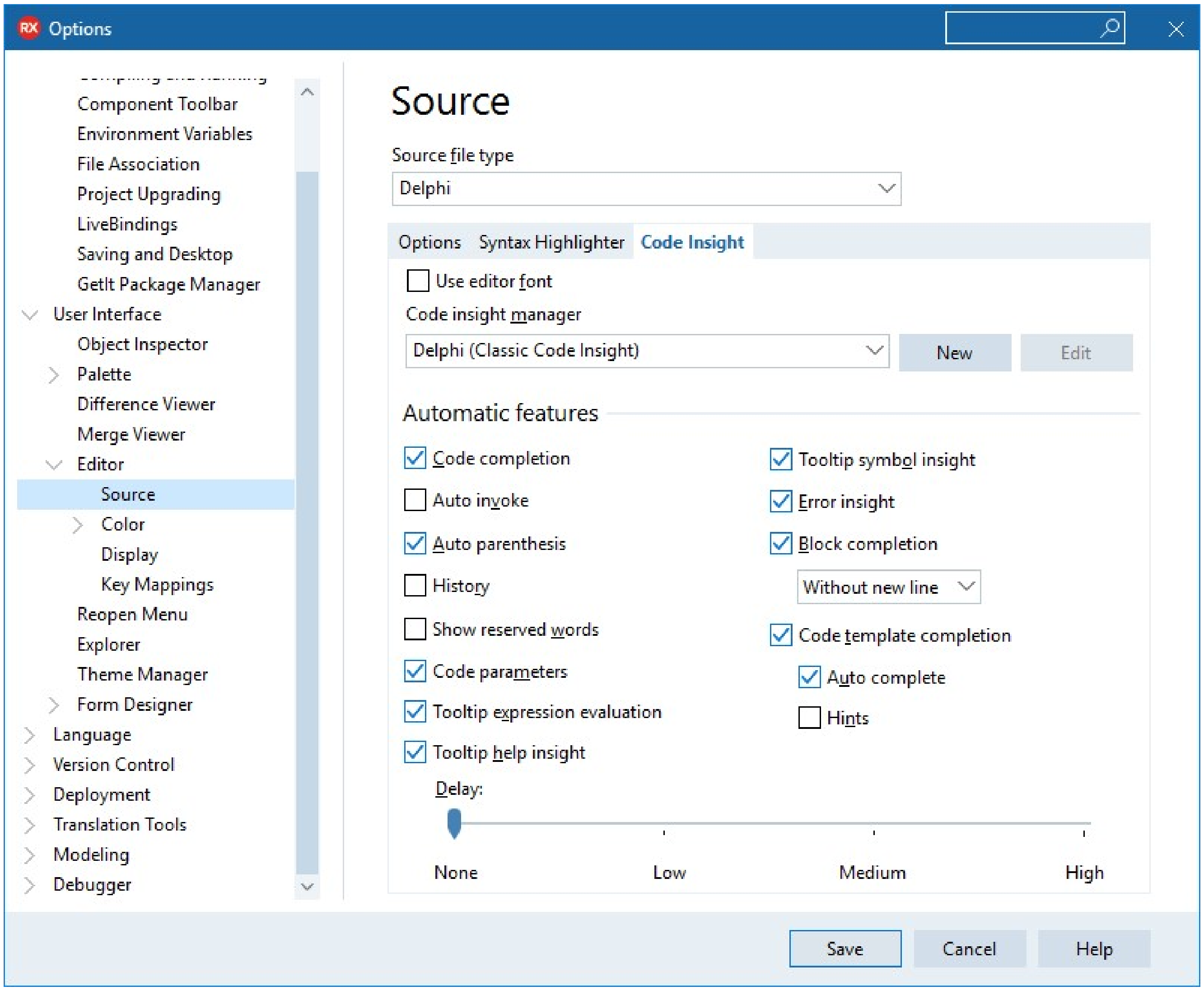
For Delphi, you can choose the Code Insight Manager, Classic or LSP.

For C++, you cannot change it. It uses the Classic or LSP for Classic or Clang compilers.

Attention: It is not recommended to change the Code Insight manager to one that is not intended for the current language.

To change Delphi between using **LSP** and the **Classic** Code Insight implementation, choose:

- **Delphi (Language Server Protocol)**: the new LSP support.
- **Delphi (Classic Code Insight)**: the old implementation of Code Insight used in 10.3.3 and earlier. When you select this option, you turn the LSP off.



Source

Source file type: Delphi

Code Insight

- [ ] Use editor font

Code insight manager: Delphi (Classic Code Insight)

Automatic features

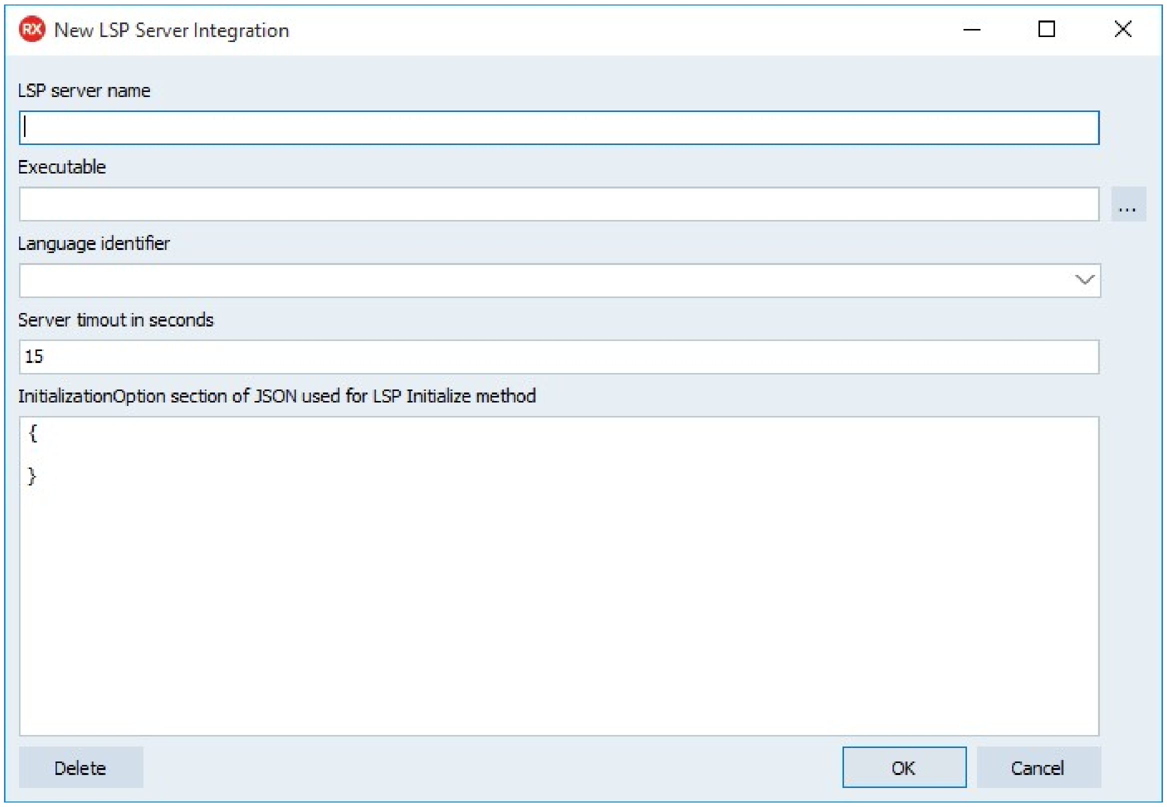
- [✓] Code completion
- [ ] Auto invoke
- [✓] Auto parenthesis
- [ ] History
- [ ] Show reserved words
- [✓] Code parameters
- [✓] Tooltip expression evaluation
- [✓] Tooltip help insight
- [✓] Tooltip symbol insight
- [✓] Error insight
- [✓] Block completion: Without new line
- [✓] Code template completion
  - [✓] Auto complete
  - [ ] Hints

Delay: None [✓] / Low / Medium / High

You can also create a **New LSP Manager**. Click on New, and a dialog appears allowing you to specify the LSP server executable and some options, including a friendly name, the language identifier, a timeout after which the server will be forcibly restarted, and any server-specific initialization options to be injected into the Initialize JSON RPC call.

You can use this option to add a server for Python or any other language.

> *Note: RAD Studio only supports servers that communicate over standard I/O (console I/O).*


New LSP Server Integration

LSP server name

Executable

Language identifier

Server timout in seconds

15

InitializationOption section of JSON used for LSP Initialize method

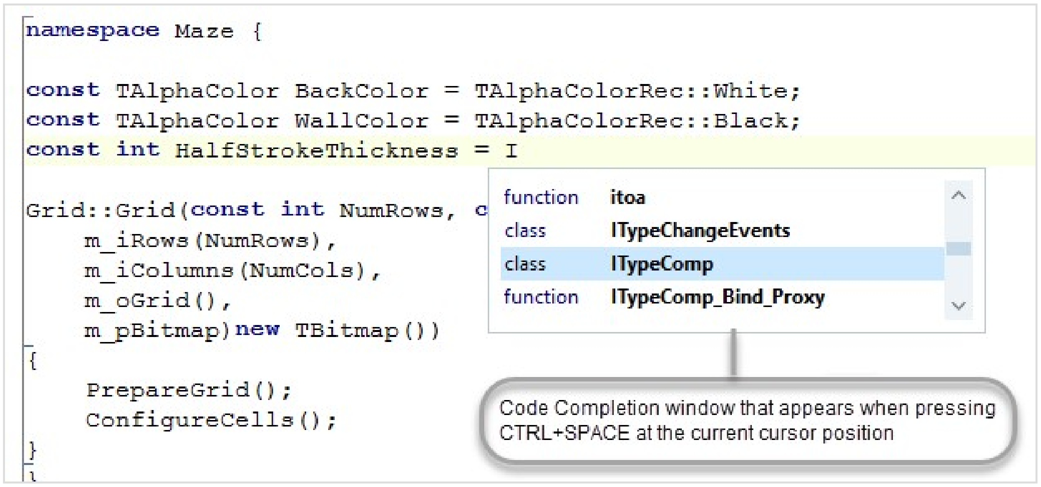
```
{
}
```

Code Completion — Ctrl+Space

The Code Completion feature displays a drop-down list of all items that contain the typed string anywhere in an identifier. To invoke Code Completion, press `Ctrl+Space` while using the Code Editor. Select the character and press `Enter` to insert the text in the code at the cursor location.

You invoke Code Completion for your specific language in the following way:

- C++
  - Press `Ctrl+Space` (always invokes Code Completion).
  - Enter `.` or `->` (only works when Auto Invoke is enabled on the Code Insight page).
  - To cancel a Code Completion request, press the `Esc` key.



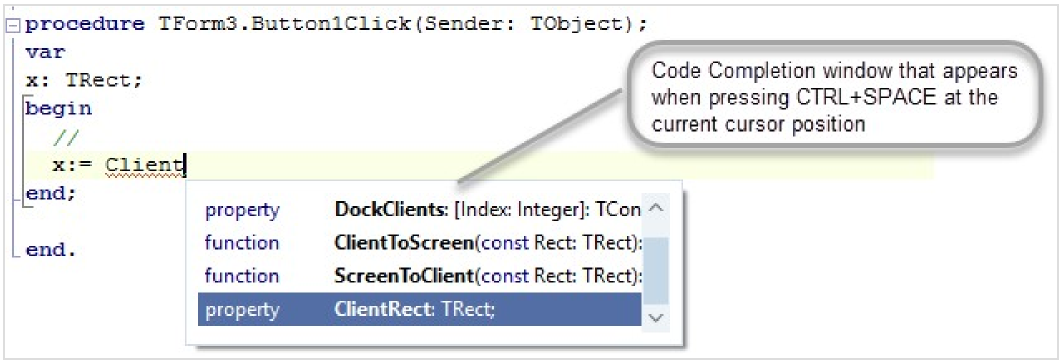
```
namespace Maze {

const TAlphaColor BackColor = TAlphaColorRec::White;
const TAlphaColor WallColor = TAlphaColorRec::Black;
const int HalfStrokeThickness = I

Grid::Grid(const int NumRows, c
    m_iRows(NumRows),
    m_iColumns(NumCols),
    m_oGrid(),
    m_pBitmap)new TBitmap())
{
    PrepareGrid();
    ConfigureCells();
}
```

function	itoa
class	ITypeChangeEvents
class	ITypeComp
function	ITypeComp_Bind_Proxy

Code Completion window that appears when pressing CTRL+SPACE at the current cursor position

> *Note: For the Clang-enhanced compiler, this feature uses a Language Server Protocol server. It is asynchronous and non-blocking.*

- `Delphi`
  - Press `Ctrl+Space` (always invokes Code Completion).
  - Enter `.` (only works when Auto Invoke is enabled on the Code Insight page).
  - To cancel a Code Completion request, press the `Esc` key.



```
procedure TForm3.Button1Click(Sender: TObject);
var
x: TRect;
begin
  //
  x:= Client
end;

end.
```

Code Completion window that appears when pressing CTRL+SPACE at the current cursor position
property DockClients: [Index: Integer]: TCon
function ClientToScreen(const Rect: TRect):
function ScreenToClient(const Rect: TRect):
property ClientRect: TRect;

When you type characters, the selection/match behavior as you type considers the following situations:

- When there is an ***exact*** match between what is typed and an identifier, it is selected.
- Otherwise, the first identifier that ***starts*** with what was typed is selected.
- Otherwise, the first identifier that ***contains*** what was typed is selected.

For example, when you type *MyControl* in the editor, invoke code completion, and then when you type *rect*, not only the properties or methods beginning with Rect are shown, but also ClientRect, BoundsRect, etc. Anything containing *rect* inside the identifier string are shown.

Code completion works correctly, including correct results, in situations including:

- A new file that does not yet exist on disk.
- A new project where neither the project nor any units exist on disk.
- A modified unit, not yet saved, where the current in-IDE code has changes that affect completion results.
- Results are correct (and changed correctly) when changing target platforms or other project options.

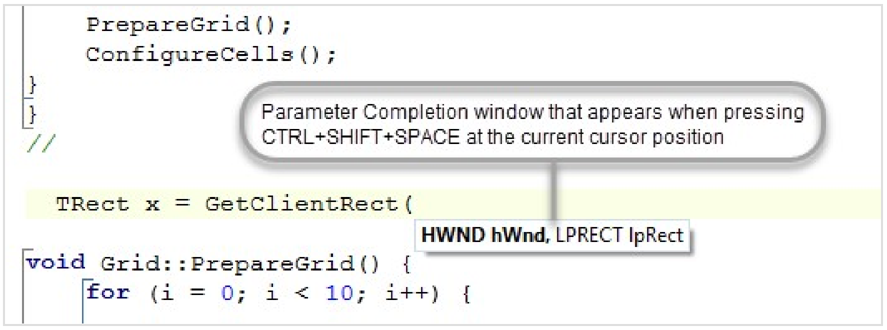
> *Note:*
>
> - *Matching is case-insensitive.*
> - *Code completion does not work inside anonymous methods.*

Parameter Completion — Ctrl+Shift+Space

It is auto-invoked after opening a left bracket of a function call, or pressing `Ctrl+Shift+Space` while using the Code Editor. A popup window displays a hint containing argument names and types for method calls. You can type characters to match those in the selection and press `Return` to insert the text in the code at the cursor location.

Examples:

- `C++`

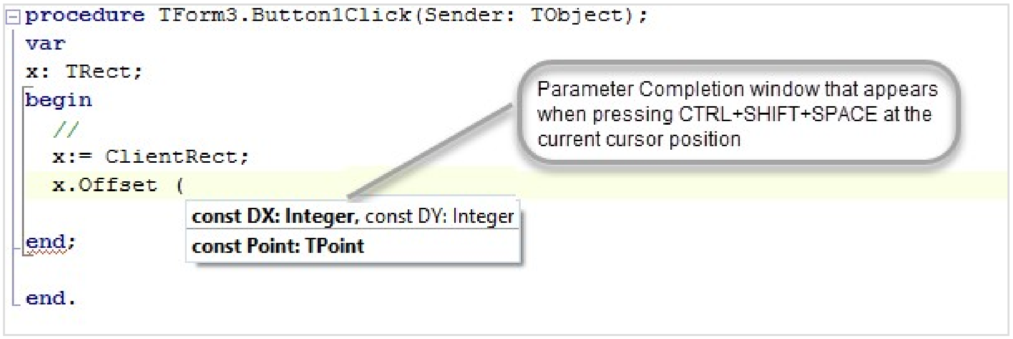


```
    PrepareGrid();
    ConfigureCells();
}
}
//

  TRect x = GetClientRect(
                              HWND hWnd, LPRECT lpRect
void Grid::PrepareGrid() {
    for (i = 0; i < 10; i++) {
```

Parameter Completion window that appears when pressing CTRL+SHIFT+SPACE at the current cursor position

Note: For the Clang-enhanced compiler, this feature uses a Language Server Protocol server. It is asynchronous and non-blocking.

- `Delphi`



```
procedure TForm3.Button1Click(Sender: TObject);
var
x: TRect;
begin
  //
  x:= ClientRect;
  x.Offset (
             const DX: Integer, const DY: Integer
end;         const Point: TPoint

end.
```

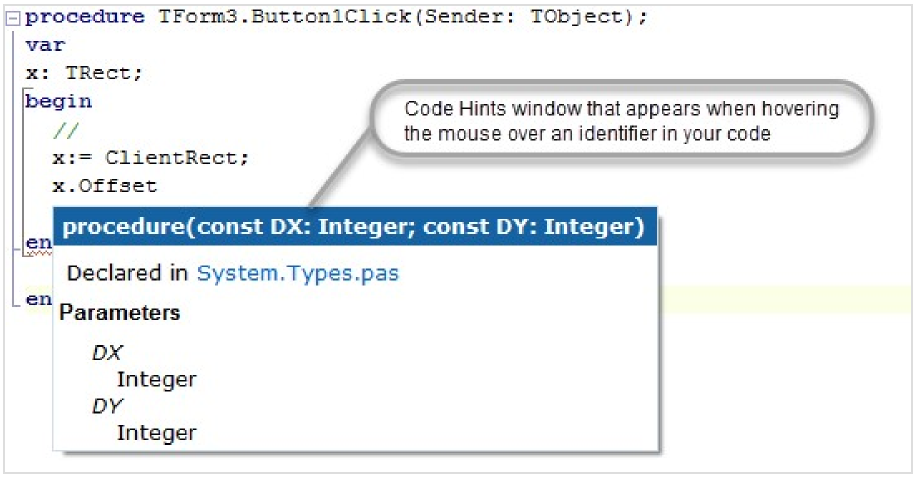
Parameter Completion window that appears when pressing CTRL+SHIFT+SPACE at the current cursor position

Code Hints

Code Hints display a hint containing information about the symbol such as type, file, and line number, where declared. You can display Code Hints by hovering the mouse over an identifier in your code, while working in the Code Editor.

Note: Code Hints only work for `Delphi` *when you have disabled the Help Insight feature. To disable Help Insight, uncheck **Tooltip help insight** on the Tools > Options > Editor Options > Code Insight dialog box.*

Example:



```
procedure TForm3.Button1Click(Sender: TObject);
var
x: TRect;
begin
  //
  x:= ClientRect;
  x.Offset
en
en
```

procedure(const DX: Integer; const DY: Integer)

Declared in System.Types.pas

Parameters

DX
Integer
DY
Integer

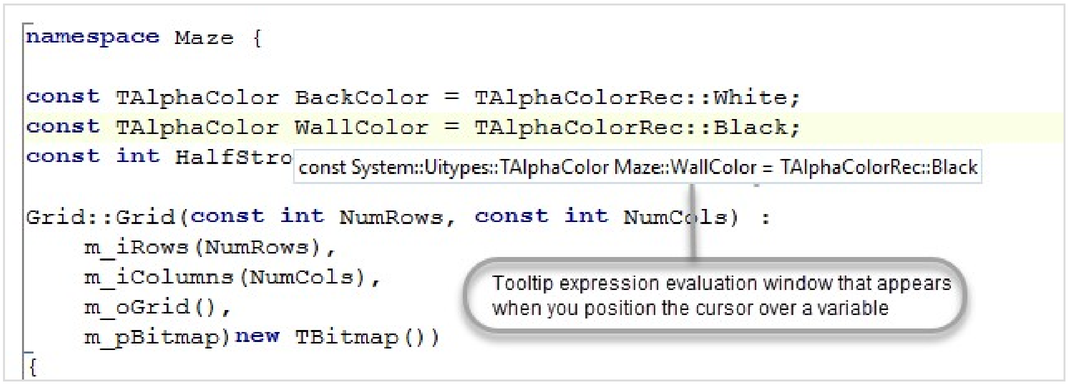
Code Hints window that appears when hovering the mouse over an identifier in your code

Tooltip expression evaluation

Displays the current value of a variable when you position the cursor over it. This feature is available when program execution is paused during a debugging session.

Examples:

- C++

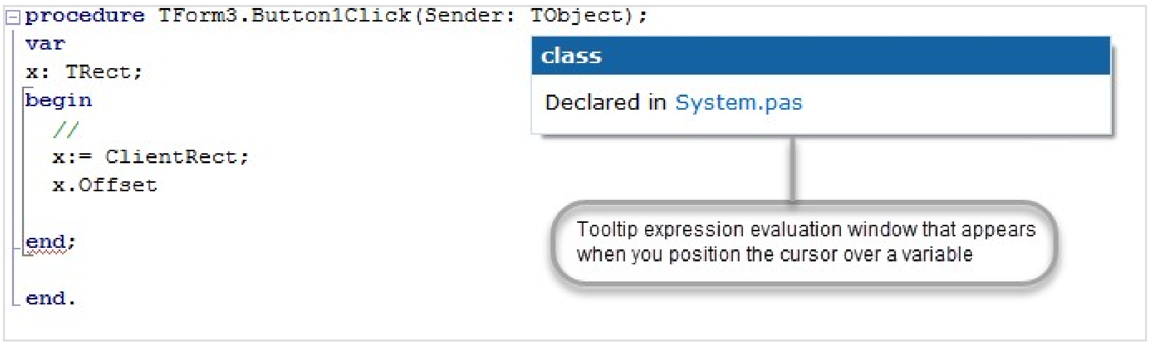


```
namespace Maze {

const TAlphaColor BackColor = TAlphaColorRec::White;
const TAlphaColor WallColor = TAlphaColorRec::Black;
const int HalfStro

Grid::Grid(const int NumRows, const int NumCols) :
    m_iRows(NumRows),
    m_iColumns(NumCols),
    m_oGrid(),
    m_pBitmap)new TBitmap())
{
```

const System::Uitypes::TAlphaColor Maze::WallColor = TAlphaColorRec::Black

Tooltip expression evaluation window that appears when you position the cursor over a variable

- Delphi



```
procedure TForm3.Button1Click(Sender: TObject);
var
x: TRect;
begin
  //
  x:= ClientRect;
  x.Offset

end;

end.
```

class

Declared in System.pas

Tooltip expression evaluation window that appears when you position the cursor over a variable

Tooltip Insight

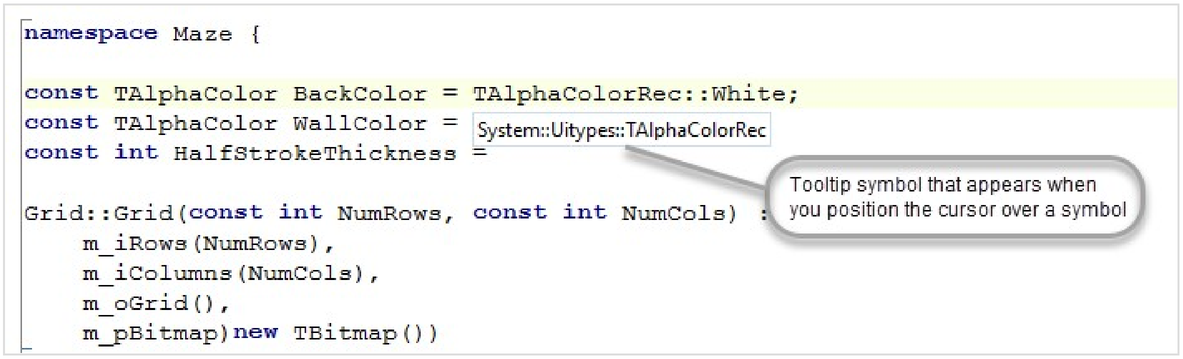
It happens when hovering the mouse over a symbol.

Tooltip Symbol Insight

Displays declaration information in a tooltip window for any identifier by passing the cursor over a symbol in the Code Editor.

Examples:

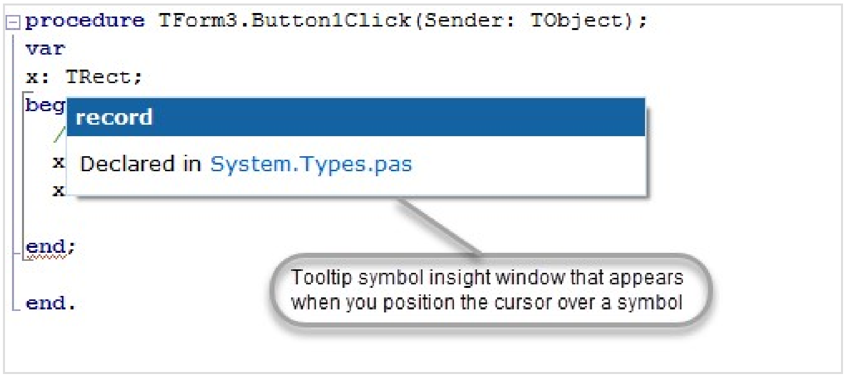
- C++



```
namespace Maze {

const TAlphaColor BackColor = TAlphaColorRec::White;
const TAlphaColor WallColor = System::Uitypes::TAlphaColorRec
const int HalfStrokeThickness =

Grid::Grid(const int NumRows, const int NumCols) :
    m_iRows(NumRows),
    m_iColumns(NumCols),
    m_oGrid(),
    m_pBitmap)new TBitmap())
```

Tooltip symbol that appears when you position the cursor over a symbol

- Delphi



```
procedure TForm3.Button1Click(Sender: TObject);
var
x: TRect;
beg
  /
  x
  x

end;

end.
```

record

Declared in System.Types.pas

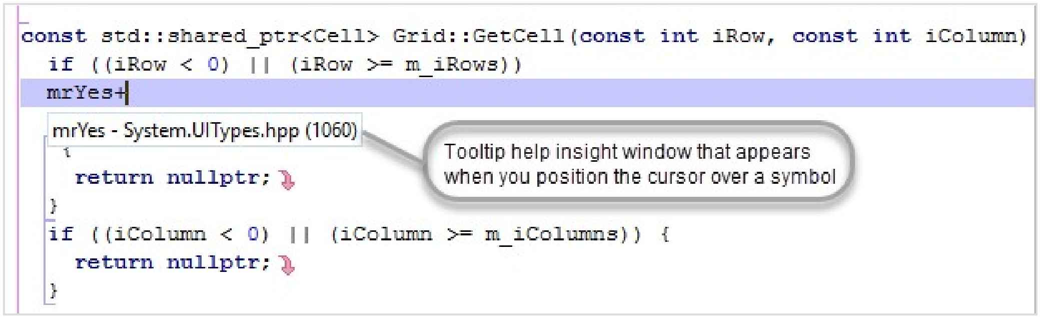
Tooltip symbol insight window that appears when you position the cursor over a symbol

Tooltip Help Insight

Displays a short description in a tooltip window when you pass the cursor over a symbol in the Code Editor.

Examples:

- C++

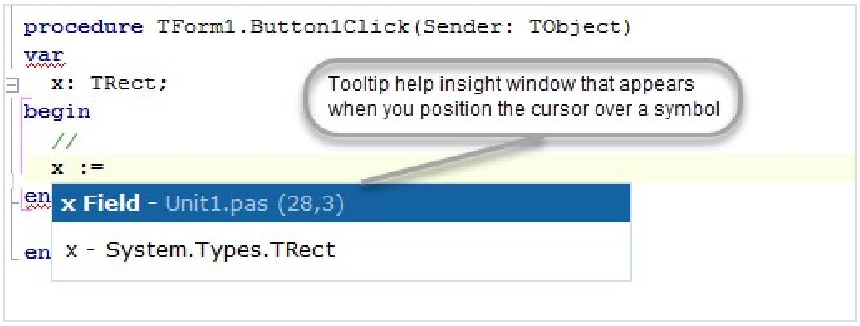


```
const std::shared_ptr<Cell> Grid::GetCell(const int iRow, const int iColumn)
  if ((iRow < 0) || (iRow >= m_iRows))
  mrYes+
  mrYes - System.UITypes.hpp (1060)
    return nullptr;
  }
  if ((iColumn < 0) || (iColumn >= m_iColumns)) {
    return nullptr;
  }
```

Tooltip help insight window that appears when you position the cursor over a symbol

Note: Only available when using the Classic Compiler

- Delphi



```
procedure TForm1.Button1Click(Sender: TObject)
var
  x: TRect;
begin
  //
  x :=
en  x Field - Unit1.pas (28,3)

en  x - System.Types.TRect
```

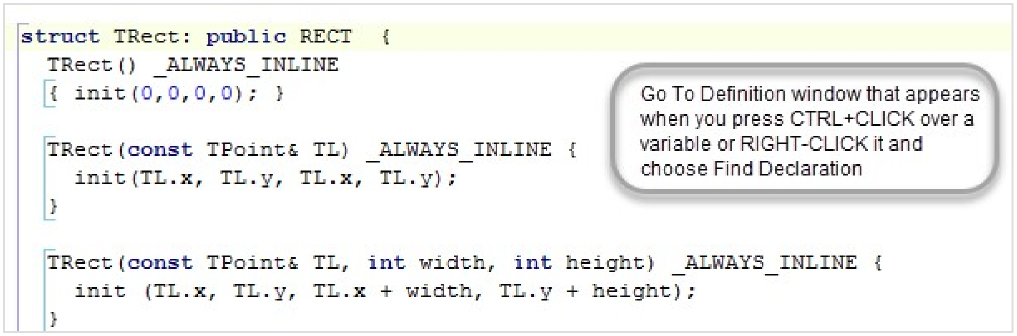
Tooltip help insight window that appears when you position the cursor over a symbol

Go To Definition

It finds where a particular type or variable was originally defined. Invoke Go To Definition by pressing Ctrl-Click over a variable or right-clicking it and choosing Find Declaration.

Examples:

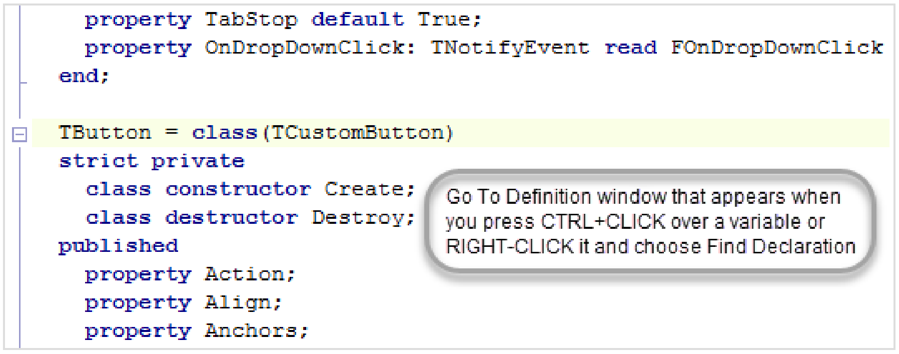
- C++



```
struct TRect: public RECT  {
  TRect() _ALWAYS_INLINE
  { init(0,0,0,0); }

  TRect(const TPoint& TL) _ALWAYS_INLINE {
    init(TL.x, TL.y, TL.x, TL.y);
  }

  TRect(const TPoint& TL, int width, int height) _ALWAYS_INLINE {
    init (TL.x, TL.y, TL.x + width, TL.y + height);
  }
```

Go To Definition window that appears when you press CTRL+CLICK over a variable or RIGHT-CLICK it and choose Find Declaration

- Delphi



```
    property TabStop default True;
    property OnDropDownClick: TNotifyEvent read FOnDropDownClick
  end;

  TButton = class(TCustomButton)
  strict private
    class constructor Create;
    class destructor Destroy;
  published
    property Action;
    property Align;
    property Anchors;
```

Go To Definition window that appears when you press CTRL+CLICK over a variable or RIGHT-CLICK it and choose Find Declaration

Block Completion — Enter key

When you press `Enter` while working in the Code Editor and there is a block of code that is incorrectly closed, the Code Editor enters the closing block token at the next available empty line after the current cursor position.

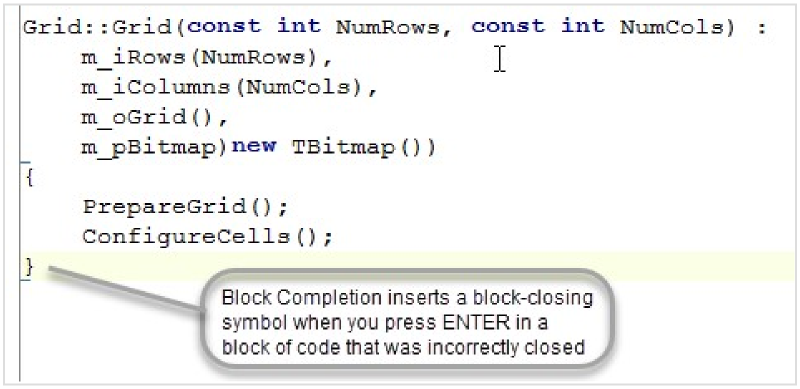
The drop-down menu sets the behavior of block completion when you surround existing statements with block symbols, as follows:

- Without new line - Positions the cursor after the block you just created.
- With new line - Positions the cursor inside the block you just created.
- New blocks only - Invokes block completion only if you start a new block.

Examples:

- C++

Start to write a function with `{` and then press `ENTER`, the Code Editor automatically completes the statement so that you have: `{ };` .

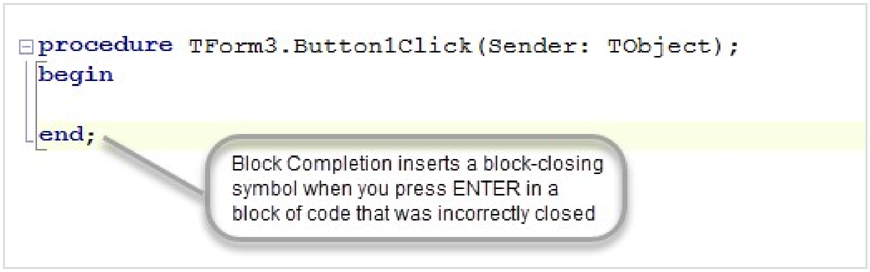


```
Grid::Grid(const int NumRows, const int NumCols) :
    m_iRows(NumRows),
    m_iColumns(NumCols),
    m_oGrid(),
    m_pBitmap)new TBitmap())
{
    PrepareGrid();
    ConfigureCells();
}
```

Block Completion inserts a block-closing symbol when you press ENTER in a block of code that was incorrectly closed

- Delphi

Type the token `begin` and then press `ENTER`, the Code Editor automatically completes the statement so that you have: `begin end;` .



```
procedure TForm3.Button1Click(Sender: TObject);
begin

end;
```

Block Completion inserts a block-closing symbol when you press ENTER in a block of code that was incorrectly closed

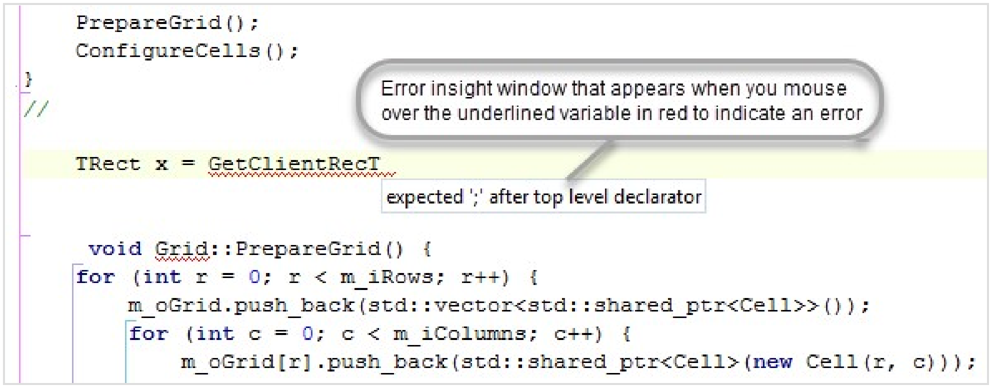
Error Insight

The Error Insight feature underlines invalid code in red. Positioning the cursor over invalid text displays a tooltip window containing the probable cause of the error.

Also, the list of errors generated by the expression appears in the Errors pane of the **Structure View**.

Examples:

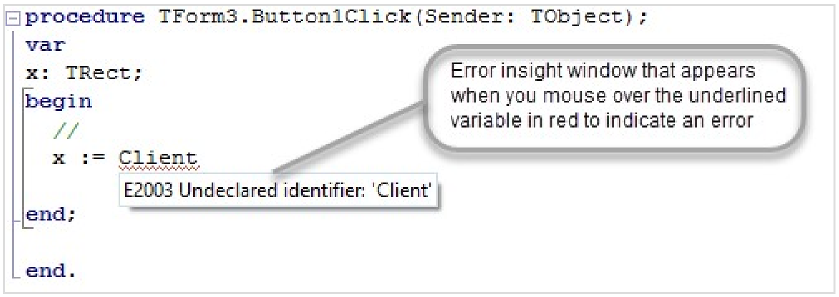
- C++



```
    PrepareGrid();
    ConfigureCells();
}
//

    TRect x = GetClientRecT
              expected ';' after top level declarator

     void Grid::PrepareGrid() {
    for (int r = 0; r < m_iRows; r++) {
        m_oGrid.push_back(std::vector<std::shared_ptr<Cell>>());
        for (int c = 0; c < m_iColumns; c++) {
            m_oGrid[r].push_back(std::shared_ptr<Cell>(new Cell(r, c)));
```

Error insight window that appears when you mouse over the underlined variable in red to indicate an error

- Delphi



```
procedure TForm3.Button1Click(Sender: TObject);
var
x: TRect;
begin
  //
  x := Client
end;

end.
```

E2003 Undeclared identifier: 'Client'

Error insight window that appears when you mouse over the underlined variable in red to indicate an error

Help Insight — Ctrl+Shift+H

Help Insight displays a hint containing information about the symbol such as type, file, line number where declared, and any XML documentation associated with the symbol (if available).

Invoke Help Insight by hovering the mouse over an identifier in your code, while working in the Code Editor. You can also invoke Help Insight by pressing `CTRL+SHIFT+H` .

Class Completion — Ctrl+Shift+C

Class Completion simplifies the process of defining and implementing new classes by generating skeleton code for the class members that you declare.

Position the cursor within a class declaration in the interface section of a unit and press `Ctrl+Shift+C` . Any unfinished property declarations are completed.

For any methods that require an implementation, empty methods are added to the implementation section.

Class Completion can also be achieved by choosing the option Complete class at cursor from the Code Editor context menu.

Code Browsing — Ctrl+Click

While using the Code Editor you can use `Ctrl+click` to automatically "jump to" the code that defines an identifier. To browse code, hold down the `Ctrl` key while hovering the mouse over the name of any class, variable, property, method, or other identifier.

The mouse pointer turns into a hand, and the identifier appears highlighted and underlined. Click the highlighted identifier, and the Code Editor jumps to the declaration of the identifier, opening the source file, if necessary. You can do the same thing by right-clicking an identifier and choosing **Find Declaration**. Pressing `Alt+Left Arrow` returns you to where you browsed from.

Code browsing can find and open only units in the project **Search path** or **Source path**, or in the product **Browsing path** or **Library path**. Directories are searched in the following order:

1. The project Search path (Delphi) or Include path (C++)
2. The project Source path, defined as the directory in which the project was saved
3. The global Browsing path
4. The global Library path
5. The Library path, which is searched only if there is no project open in the IDE

These paths can be modified by editing the corresponding values:

- Either the project-specific **Search path** for Delphi (**Project > Options > Delphi Compiler**) or the **Include path** for C++ (**Project > Options > C++ (Shared Options)**).
- The global **Browsing path** and the **Library path**:

    - For Delphi: **Tools > Options > Environment Options > Delphi Options > Library**
    - For C++: **Tools > Options > Environment Options > C++ Options > Paths and Directories**
- The global library path is set on the Add Runtime Package dialog box.

An alternative to using `Ctrl+click` to go to the declaration of an identifier is to use `Alt+Up` (arrow). For other useful key combinations, see Default Keyboard Shortcuts.

Advanced Options

Index for C++ Insights

Provides index for the tooltip insight and go to definition features.

It is the LSP server (cquery) that runs through the project and builds a database. It uses CPU, so it can be turned off through the Index for C++ Insights checkbox.

Tip: Keep in mind that if this option is turned off, both features stop working.

Note: For the Clang-enhanced compiler, this feature uses a Language Server Protocol server. It is asynchronous and non-blocking.

Tools API Support

Code Insight support is mainly focused around the Language Server Protocol, and there is a generic LSP support, where you can create a New LSP server using the above dialog. However, the IDE implements two things: generic asynchronous code insight; and then LSP support, which is a specific implementation of asynchronous code insight.

The ToolsAPI defines a number of new types and interfaces for async code insight. The LSP uses these interfaces, but you can write a code insight provider that is asynchronous with any implementation you wish.

The interfaces are located in ToolsAPI.pas:

- `IOTAAsyncCodeInsightManager` is the main interface for a generic async code insight manager implementation. It uses a number of callbacks, which are defined above it.
- `IOTACodeInsightUIOverride` is used to override specific UI behavior.

Filing Bugs and Log Files

To enable log files, open the registry:

- Create a key:

```
HKEY_CURRENT_USER\Software\Embarcadero\BDS\21.0\LSP
```

- Create a `DWORD` value called `DelphiLSPLog` with value hex `$ff` or decimal `255`

Log files will be located in C:\Users\<User>\AppData\Local\Temp\DelphiLSP.

Make sure you include the log files with every bug report in the Quality Portal. They are extremely helpful to track down the cause of issues.

Next

Refactoring