Manual do KDevelop

Esta documentação foi convertida a partir da Base de Utilizadores do KDE, em concreto da página KDevelop4/Manual.
Tradução: José Pires

Conteúdo

Resumo

O KDevelop é um Ambiente de Desenvolvimento Integrado para ser usado numa grande variedade de tarefas de programação.

Capítulo 1

O que é o KDevelop?

O KDevelop é um ambiente de desenvolvimento integrado (IDE) para o C++ (e outras linguagens) e que é uma das muitas aplicações do KDE. Como tal, executa-se em Linux® (mesmo que execute um dos outros ambientes de trabalho, como o GNOME) mas também está disponível para outras variantes do UNIX® e para o Windows.

O KDevelop oferece todas as capacidades dos IDE's modernos. Para grandes projectos e aplicações, a funcionalidade mais importante é que o KDevelop *compreenda o C++*: ele processa toda a base de código e recorda todas as funções-membro das classes, onde são definidas as variáveis, quais são os seus tipos, entre muitas outras coisas acerca do seu código. Por exemplo, imaginemos que um dos ficheiros de inclusão do seu projecto declara uma classe

```
class Carro {
  // ...
  public:
    std::string cor () const;
};
```

e depois no seu programa tiver

```
Carro o_meu_carro;
// ...fazer alguma coisa com essa variável...
std::string cor = o_meu_carro.co
```

terá recordado que o `o_meu_carro` da última linha é uma variável do tipo `Carro` e oferecer-se-á para completar o `co` como `cor()`, dado que esta é a única função-membro da classe `Carro` que começa desta forma. Em vez de continuar a escrever, poderá carregar em **Enter** para obter a palavra completa; isto poupa a escrita, os erros e não necessita que você recorde os nomes exactos das centenas ou milhares de funções e classes que compõem os grandes projectos.

Como segundo exemplo, assuma que tem código como o seguinte:

```
double xpto ()
{
  double var = funcao();
 return var * var;
}
double xpto2 ()
{
  double var = funcao();
  return var * var * var;
}
```

Se passar o rato sobre o símbolo `var` na função `xpto2`, irá obter uma opção para ver todos os usos deste símbolo. Se carregar nele, só irá mostrar os usos desta variável na função `xpto2`, porque o KDevelop compreende que a variável `var` na função `xpto` não tem nada a ver com ela. Da mesma forma, se carregar com o botão direito no nome da variável, poderá mudar o nome da mesma; se o fizer, só irá tocar na variável em `xpto2`, mas não em `xpto`.

Mas o KDevelop não é apenas um editor de código inteligente; existem outras coisas que o KDevelop faz bem. Obviamente, ele realça o código-fonte com diferentes cores; tem uma indentação personalizada; tem uma interface integrada com o depurador gdb da GNU; pode-lhe mostrar a documentação de uma função se passar o rato sobre um uso desta função; poderá lidar com diferentes tipos de ambientes de compilação e compiladores (isto é com o **make** e o **cmake**), entre muitas outras coisas boas que serão discutidas neste manual.

Capítulo 2

Sessões e projectos: As bases do KDevelop

Nesta secção, iremos passar por alguma da terminologia de como o KDevelop vê o mundo e como ele estrutura o trabalho. Em particular, iremos introduzir o conceito de *sessões* e *projectos*, assim como explicar como poderá configurar os projectos com que deseja trabalhar no KDevelop.

2.1 Terminologia

O KDevelop tem o conceito de *sessões* e *projectos*. Uma sessão contém todos os projectos que possam ter alguma coisa a ver entre si. Para os exemplos que se seguem, assuma que é um programador de uma biblioteca e de uma aplicação que a usa. Poderá pensar nas bibliotecas de base do KDE como o primeiro caso e o KDevelop como o segundo. Outro exemplo: imagine que é um programador do 'kernel' do Linux® mas que está também a trabalhar num controlador de dispositivo do Linux® que ainda não foi reunido com a árvore de código do 'kernel'.

Como tal, pegando no último exemplo, teria uma sessão no KDevelop com dois projectos: o 'kernel' do Linux® e o controlador do dispositivo. Irá querer agrupá-los numa única sessão (em vez de ter duas sessões com um projecto em cada uma), porque será útil poder ver as funções e estruturas de dados do 'kernel' no KDevelop sempre que escrever código para o controlador — por exemplo, para que possa ver os nomes das funções e variáveis do 'kernel' automaticamente expandidas ou para que possa ver a documentação das funções do 'kernel' enquanto programa o controlador.

Agora imagine que também é um programador do KDE. Então iria ter uma segunda sessão que tivesse o KDe como um projecto. Poderia em princípio ter apenas uma sessão para tudo isto, mas não existe nenhuma razão real para tal: no seu trabalho com o KDE, não precisa de aceder às funções do 'kernel' ou do controlador de dispositivos; da mesma forma, também não irá querer os nomes das classes do KDE expandidos automaticamente quando estiver a trabalhar no 'kernel' do Linux®. Finalmente, a compilação de algumas das bibliotecas do KDE é independente da recompilação do 'kernel' do Linux® (embora, contudo, seja normal compilar o 'kernel' do Linux® quando estiver a compilar o controlador do dispositivo, caso alguns dos ficheiros do 'kernel' tenham mudado).

Finalmente, outro uso para as sessões é se trabalhar tanto na versão actualmente em desenvolvimento de um projecto como noutra versão em paralelo: nesse caso, não irá querer que o KDevelop confunda as classes que pertencem à versão principal com as da alternativa, pelo que terá duas sessões com o mesmo conjunto de projectos, mas com pastas diferentes (correspondendo a diferentes ramificações de desenvolvimento).

2.2 Configurar uma sessão e importar um projecto existente

Vamos continuar com o exemplo do 'kernel' do Linux® e do controlador do dispositivo — poderá querer substituir o seu conjunto próprio de bibliotecas ou projectos para estes dois exemplos. Para criar uma nova sessão que contenha estes dois projectos, vá a **Sessão** → **Iniciar uma nova sessão** no cimo e à esquerda (ou, se for a primeira vez que usar o KDevelop: basta usar a sessão predefinida que obtém na primeira utilização, que está vazia).

Iremos querer preencher esta sessão com projectos que, para já, assumimos que já existem em algum lado (o caso de iniciar os projectos do zero é discutido noutro ponto do manual). Para tal, existem essencialmente dois métodos, dependendo se o projecto já existe em algum lado do seu disco ou se precisa de ser transferido a partir de um servidor.

2.2.1 Opção 1: Importar um projecto de um sistema de controlo de versões

Iremos assumir que o projecto que desejamos configurar — o 'kernel' do Linux® — reside nalgum sistema de controlo de versões num servidor, mas que ainda não o extraiu para o seu disco rígido local. Nesse caso, vá ao menu **Projecto** para criar o 'kernel' do Linux® como um projecto dentro da sessão actual e depois siga estes passos:

- Vá a **Projectos** → **Obter o projecto** para importar um projecto
- Irá ter várias opções para iniciar um projecto novo na sessão actual, dependendo de onde vêm os ficheiros de origem: poderá simplesmente indicar ao KDevelop uma pasta existente (veja a opção 2 abaixo) ou poderá pedir ao KDevelop para obter a listagem de um repositório.
- Assumindo que não tem já uma versão extraída do servidor:
  - Na janela, em **Seleccionar a origem**, opte por usar o **Subversion**, **Git** ou alguma das outras escolhas
  - Escolha uma pasta de trabalho como destino para onde será extraído o código
  - Escolha o URL da localização no repositório onde se podem obter os ficheiros de código
  - Carregue em **Obter**. Isto poderá levar bastante tempo, dependendo da velocidade da sua ligação e do tamanho do projecto. Infelizmente, no KDevelop 4.2.x, a barra de progresso não mostra nada de facto, mas você poderá seguir a evolução se olhar periodicamente para o resultado do comando da consola

    ```
    du -sk /local/do/projecto/do/KDevelop
    ```

    para ver quantos dados já foram transferidos.

> NOTA
> O problema com a barra de progresso foi comunicado como sendo o erro 256832 do KDevelop.

> NOTA
> Neste processo, obtenho também a mensagem de erro *Tem de indicar uma localização válida para o projecto*; esta poderá ser ignorada sem problemas.

- Pede-lhe para seleccionar um ficheiro de projecto do KDevelop nesta pasta. Dado que provavelmente não terá ainda nenhum definido, basta carregar em **Seguinte**

- Carregue em **Seguinte** de novo
- O KDevelop então pedir-lhe-á para escolher um gestor do projecto. Se este projecto usar os ficheiros do 'make' do UNIX®, escolha o gestor de projectos com 'makefiles' personalizadas
- O KDevelop irá então começar a processar o projecto inteiro. Mais uma vez, irá levar bastante tempo a percorrer todos os ficheiros e a indexar as classes, etc. Na parte inferior direita da janela principal, existe uma barra de progresso que mostra quanto é que este processo já percorreu (se tiver vários processadores, poderá acelerar este processo se for à opção **Configuração** → **Configurar o KDevelop** e seleccionar o **Processador em segundo plano** à esquerda, aumentando o número de tarefas de processamento em segundo plano à direita.)

2.2.2 Opção 2: Importar um projecto que já exista no seu disco rígido

Em alternativa, se o projecto com que deseja trabalhar já existir no seu disco rígido (por exemplo, porque o transferiu como um ficheiro 'tar' de um servidor de FTP, porque já obteve uma versão do projecto a partir de um sistema de controlo de versões ou porque é o seu próprio projecto existente *apenas* no seu próprio disco rígido), então use **Projectos** → **Abrir/Importar um Projecto** e, na janela que aparece, escolha a pasta onde se encontra o seu projecto.

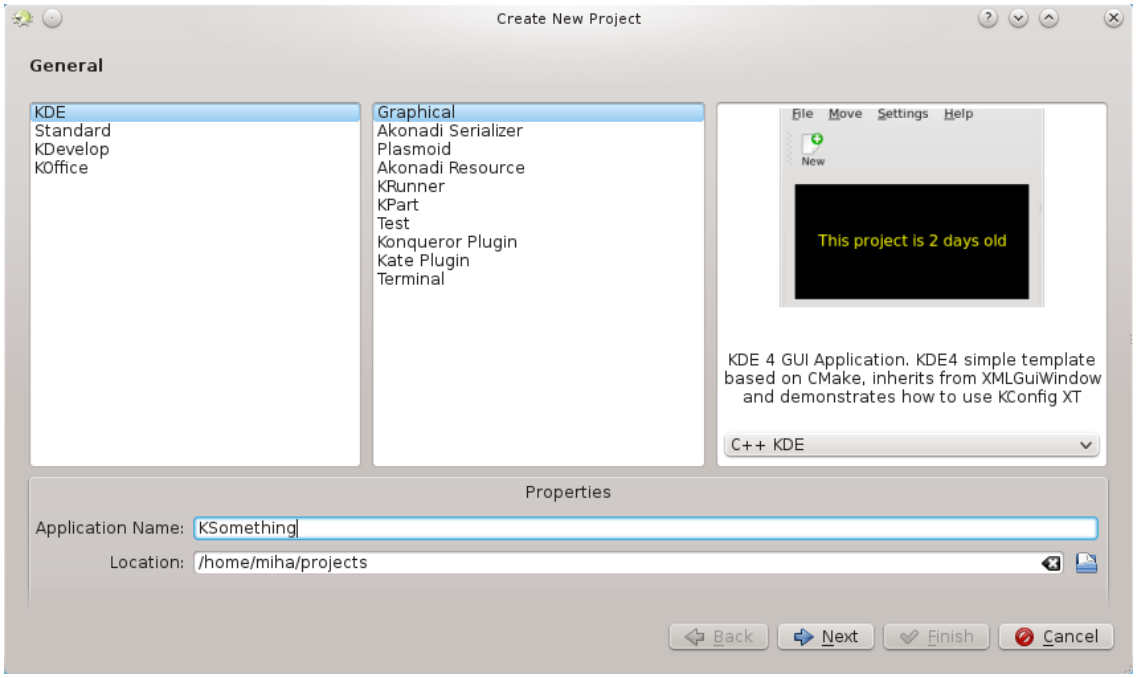
2.3 Configurar uma aplicação como um segundo projecto

A próxima acção que quererá fazer é configurar outros projectos na mesma sessão. No exemplo acima, poderá querer adicionar o controlador do dispositivo como segundo projecto, o qual poderá fazer usando exactamente os mesmos passos.

Se tiver várias aplicações ou bibliotecas, basta repetir os passos para adicionar cada vez mais projectos à sua sessão.

2.4 Criar projectos do zero

Existe obviamente também a possibilidade de iniciar um novo projecto do zero. Isso pode ser feito se usar a opção do menu **Projectos** → **Novo a Partir de Modelo...**, que lhe apresenta uma janela de selecção de modelos. Alguns modelos de projectos são fornecidos com o KDevelop, mas estão disponíveis ainda mais se instalar a aplicação **KAppTemplate**. Escolha o tipo de projecto e a linguagem de programação na janela, indique um nome e local para o seu projecto e carregue em **Seguinte**.



A segunda página da janela permite-lhe configurar um sistema de controlo de versões. Escolha o sistema que deseja usar e preencha a configuração específica do sistema, caso necessário. Se não quiser usar um sistema de controlo de versões ou se o quiser configurar manualmente, escolha **Nenhum**. Quando estiver satisfeito com a sua escolha, carregue em **Terminar**.

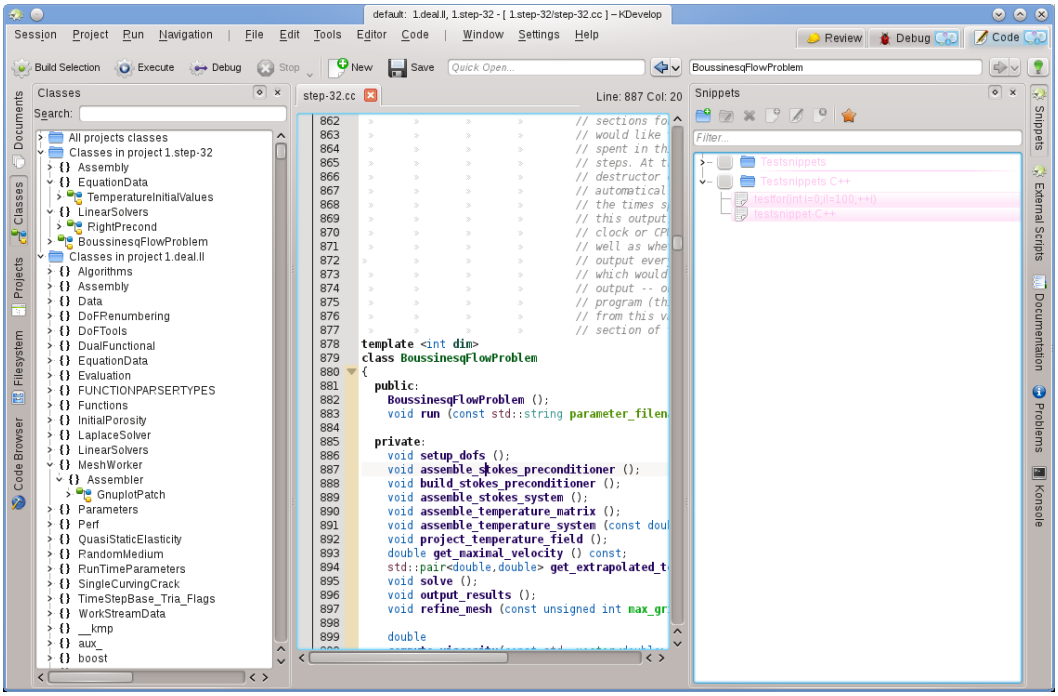
O seu projecto está agora criado, pelo que poderá tentar compilá-lo ou instalá-lo. Alguns modelos irão incluir comentários dentro do código, ou até mesmo um ficheiro README (leia-me) separado, sendo recomendado que os leia em primeiro lugar. Depois, poderá começar a trabalhar no seu projecto, adicionando as funcionalidades que desejar.

Capítulo 3

Lidar com o código-fonte

Para além da depuração, da leitura e escrita de código, será onde irá gastar mais tempo a desenvolver as aplicações. Para tal, o KDevelop oferece-lhe várias formas de explorar o código-fonte e de tornar mais produtiva a escrita do mesmo. Como será discutido com mais detalhe nas seguintes secções, o KDevelop não é apenas um editor de código — em vez disso, é um sistema de gestão de código que lhe dá diferentes vistas sobre a informação extraída dos ficheiros que compõem o código-fonte da sua sessão.

3.1 Ferramentas e janelas



Para lidar com os projectos, o KDevelop tem o conceito de *ferramentas*. Uma ferramenta oferece uma vista em particular sobre o código ou uma acção a efectuar com ele. As ferramentas são representadas como botões em torno da sua janela (com texto vertical ao longo das margens esquerda e direita ou ainda a nível horizontal, ao longo da margem inferior). Se carregar nelas,

as mesmas expandir-se-ão para uma sub-janela — uma *área ou vista* — dentro da janela principal; se carregar no botão de ferramentas de novo, a sub-janela desaparece de novo.

Para fazer desaparecer uma sub-janela, também poderá carregar no **x** no canto superior-direito da sub-janela

A imagem acima mostra uma selecção em particular das ferramentas, alinhadas ao longo das margens esquerda e direita; na imagem, a ferramenta de **Classes** está aberta à esquerda e os **Excertos** à direita, em conjunto com um editor de um ficheiro de código no meio. Na prática, na maior parte do tempo irá ter provavelmente apenas o editor e talvez a ferramenta de **Classes** ou o **Navegador do Código** abertas à esquerda. As outras áreas de ferramentas provavelmente só estarão abertas temporariamente para você usar a ferramenta, deixando a maior parte do tempo o espaço livre para o editor.

Quando executar o KDevelop da primeira vez, irá ter já o botão de ferramentas de **Projectos**. Carregue nele: irá abrir uma sub-janela que mostra os projectos que tiver adicionado à sessão no fundo, assim como uma vista do sistema de ficheiros das pastas dos seus projectos no topo.

Existem muitas outras ferramentas que poderá usar com o KDevelop, onde nem todas estão inicialmente presentes como botões no perímetro. Para adicionar algumas, vá à opção do menu **Janelas** → **Adicionar uma área de ferramentas**. Aqui estão algumas que poderá achar úteis:

- **Classes**: Uma lista completa de todas as classes que estão definidas num dos projectos ou na sua sessão, com todas as suas funções e variáveis-membros. Se carregar em qualquer dos membros, irá abrir um editor de código no local do item onde carregou.
- **Documentos**: Apresenta alguns dos ficheiros visitados recentemente, classificados pelo tipo (isto é ficheiros de código, ficheiros de modificações, documentos de texto simples).
- **Navegador de Código**: Dependendo da posição do seu cursor num ficheiro, esta ferramenta mostra as coisas que estejam relacionadas entre si. Por exemplo, se estiver numa linha `#include`, irá mostrar informações acerca do ficheiro que esta a incluir, como as classes que estão declaradas nesse ficheiro; se estiver numa linha vazia ao nível do ficheiro, irá mostrar as classes e funções declaradas e definidas no ficheiro actual (tudo como hiperligações: se carregar nelas, irá para o ponto do ficheiro onde se encontra de facto a declaração ou definição); se estiver na definição de uma função, ela mostra onde se encontra a declaração e oferece uma lista dos locais onde é usada a função.
- **Sistema de ficheiros**: Mostra-lhe uma vista em árvore do sistema de ficheiros.
- **Documentação**: Permite-lhe procurar nas páginas de manual e noutros documentos de ajuda.
- **Excertos**: Isto fornece sequências de texto que uma pessoa poderá usar quantas vezes quiser e que não terá de escrever sempre. Por exemplo, no projecto em que foi criada a imagem acima, existe uma necessidade frequente de escrever código do tipo

```
for (nome-tipo Triangulacao< dim>::active_cell_iterator celula
      = triangulacao.begin_active();
   celula != triangulacao.end();
   ++celula)
```

Esta é uma expressão estranha mas terá quase sempre este aspecto sempre que precisar de um ciclo — o que a tornará um bom candidato para um excerto.

- **Konsole**: Abre uma janela de linha de comandos dentro da janela principal do KDevelop, para o comando ocasional que possa querer introduzir (isto é para executar o `./configure`).

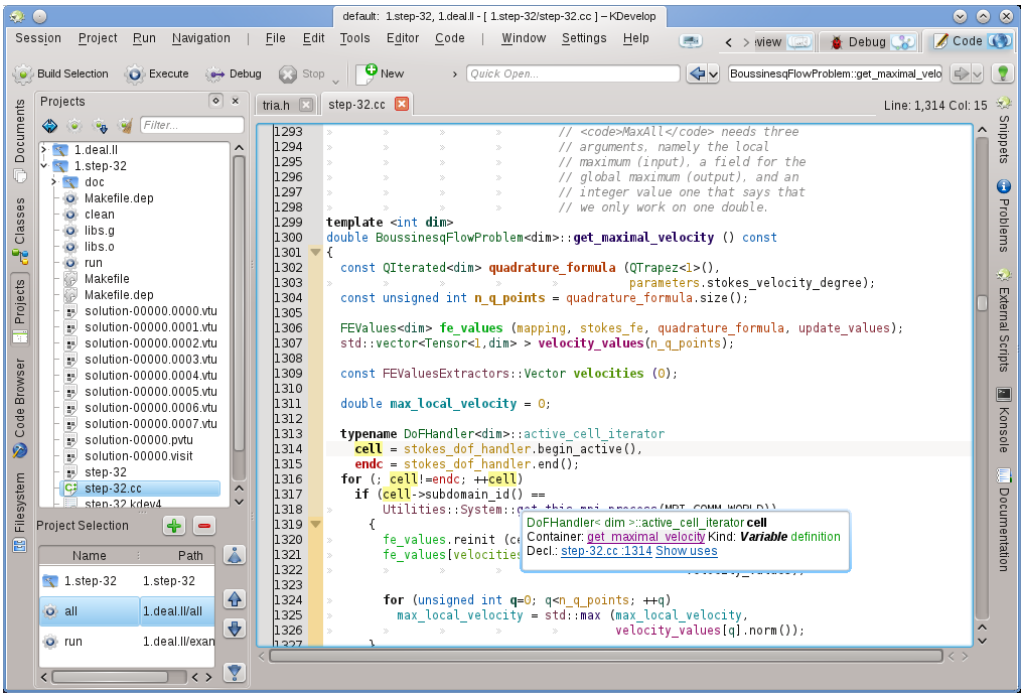
Está descrita uma lista completa das ferramentas e janelas aqui.

Para muitos programadores, o espaço vertical do ecrã é o mais importante. Para esse fim, poderá organizar as suas áreas de ferramentas nas margens esquerda e direita da janela: para mover uma ferramenta, carregue no seu símbolo com o botão direito do rato e seleccione uma posição nova para ele.

3.2 Explorar o código-fonte

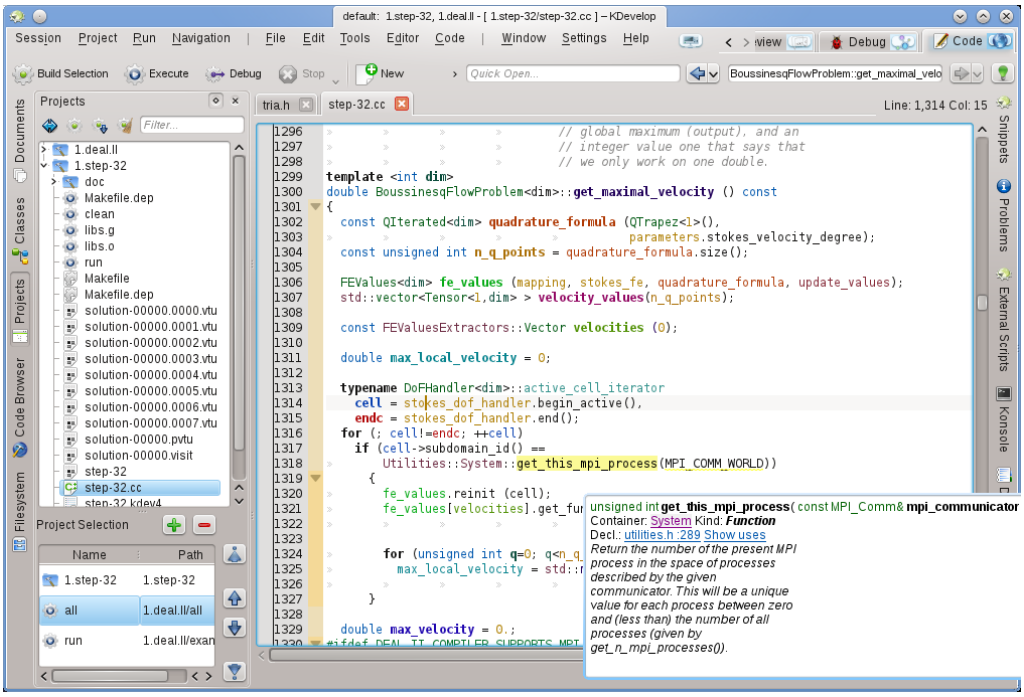
3.2.1 Informação local

O KDevelop *compreende* o código-fonte e, por consequência, é bastante bom a dar-lhe informações acerca das variáveis e funções que possam aparecer no seu programa. Por exemplo, aqui está uma imagem onde está a lidar com um pedaço de código e, ao passar o rato sobre o símbolo `celula` na linha 1316 (se estiver a trabalhar com base no teclado, poderá obter o mesmo efeito se mantiver a tecla **Alt** carregada durante um pouco):



O KDevelop mostra uma dica que inclui o tipo da variável (aqui: `DoFHandler<dim>active_cell_iterator`), onde está declarada esta variável (o *contentor*, que é aqui a função envolvente `velocidade_maxima`, dado que é uma variável local), o que é (uma variável, não uma função, classe ou espaço de nomes) e onde está declarada (na linha 1314, umas linhas acima no código).

No contexto actual, o símbolo sobre o qual o rato passou não tinha documentação associada. Nesse exemplo, se o rato tivesse passado sobre o símbolo `get_this_mpi_process`, na linha 1318, o resultado teria sido o seguinte:

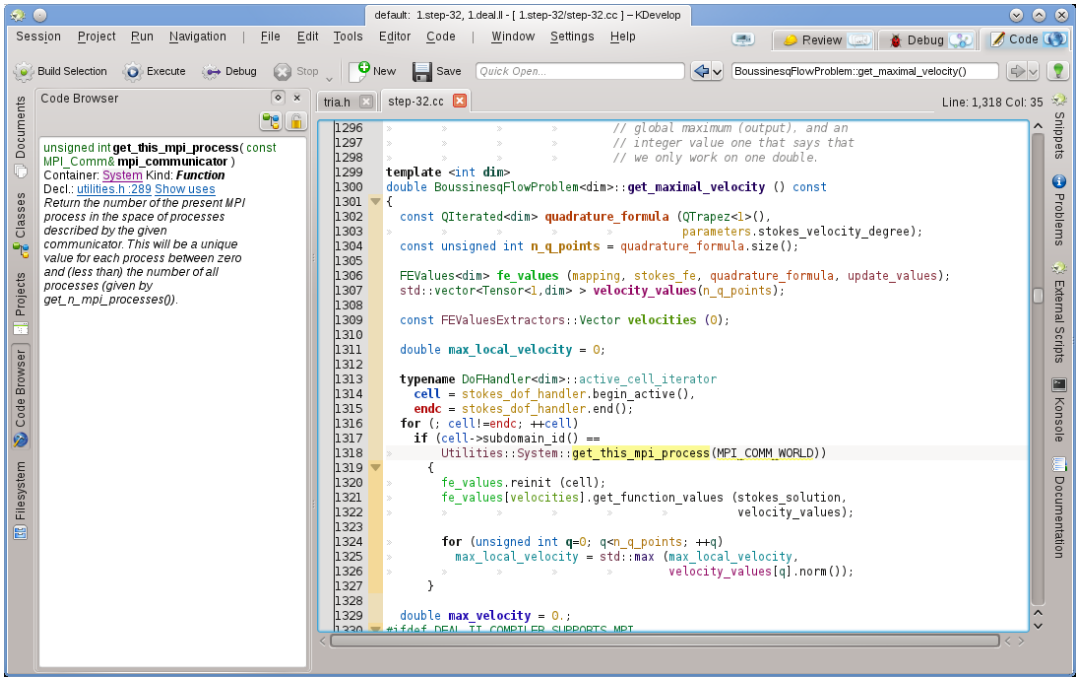


Aqui, o KDevelop cruzou a informação da declaração a partir de um ficheiro completamente diferente (o `utilities.h`, que reside de facto num projecto diferente na mesma sessão), em conjunto com o comentário do 'doxygen' que acompanha a declaração nesse local.

O que torna estas dicas ainda mais úteis é o facto de serem dinâmicas: pode carregar no contentor para obter informações acerca do contexto em que a mesma é declarada (isto é no espaço de nomes `System`, como onde está declarada, definida, usada ou qual é a sua documentação) e poderá carregar nas ligações azuis que irão repor a posição do cursor no local de declaração do símbolo (isto é em `utilities.h`, na linha 289) ou dar-lhe uma lista dos locais onde este símbolo é usado no ficheiro actual ou por todos os projectos da sessão actual. A última opção é normalmente usada se quiser explorar como, por exemplo, é usada uma função em particular num grande bloco de código.

NOTA

A informação numa dica é flutuante — depende se mantém carregada a tecla **Alt** ou se passa o rato por cima. Se quiser um local mais permanente para a mesma, abra a ferramenta do **Navegador de Código** numa das sub-janelas. Por exemplo, aqui o cursor está na mesma função que no exemplo acima e a área de ferramentas à esquerda apresenta o mesmo tipo de informação que a dica anterior:



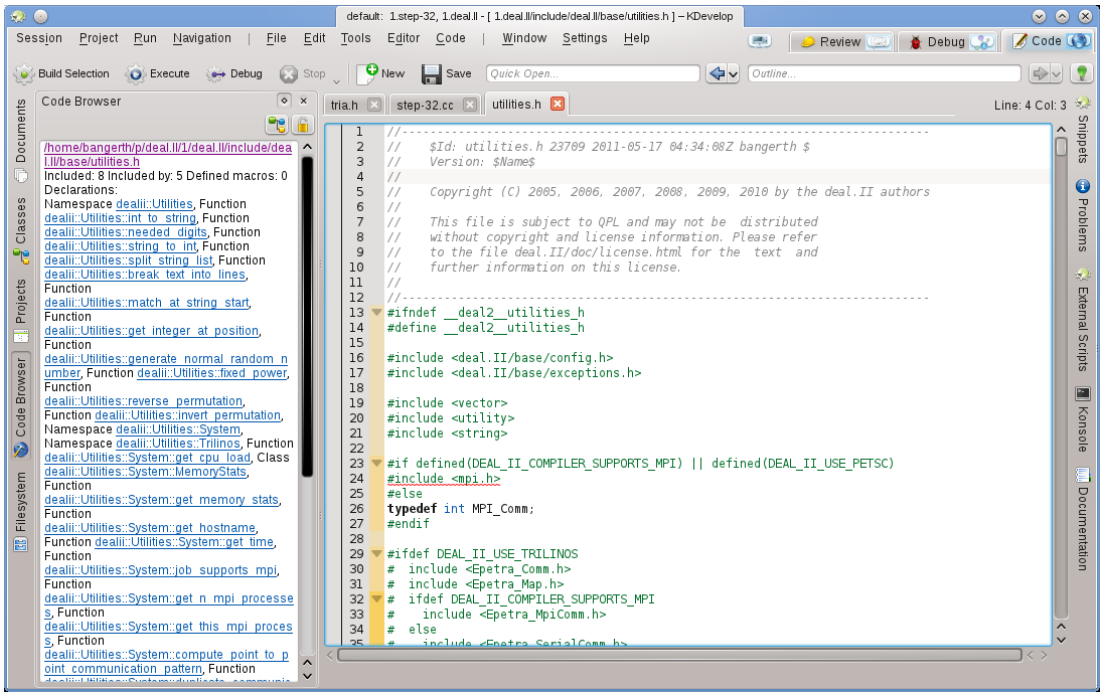
Se mover o cursor para a direita, irá mudar a informação apresentada à esquerda. Para além disso, se carregar no botão **Bloquear a janela actual**, no canto superior direito, poderá bloquear esta informação, tornando-a independente do movimento do cursor, enquanto explora a informação aí apresentada.

NOTA

Este tipo de informação de contexto está disponível em muitos outros locais no KDevelop, não apenas no editor de código. Por exemplo, se mantiver carregada a tecla **Alt** numa lista de completação (isto é ao fazer uma abertura rápida), também irá apresentar a informação de contexto do símbolo actual.

3.2.2 Informação de âmbito do ficheiro

O próximo nível acima é a obtenção de informação acerca do ficheiro de código por inteiro sobre o qual está a trabalhar. Para esse fim, coloque o cursor ao nível do ficheiro actual e veja o que a ferramenta do **Navegador de Código** irá mostrar:

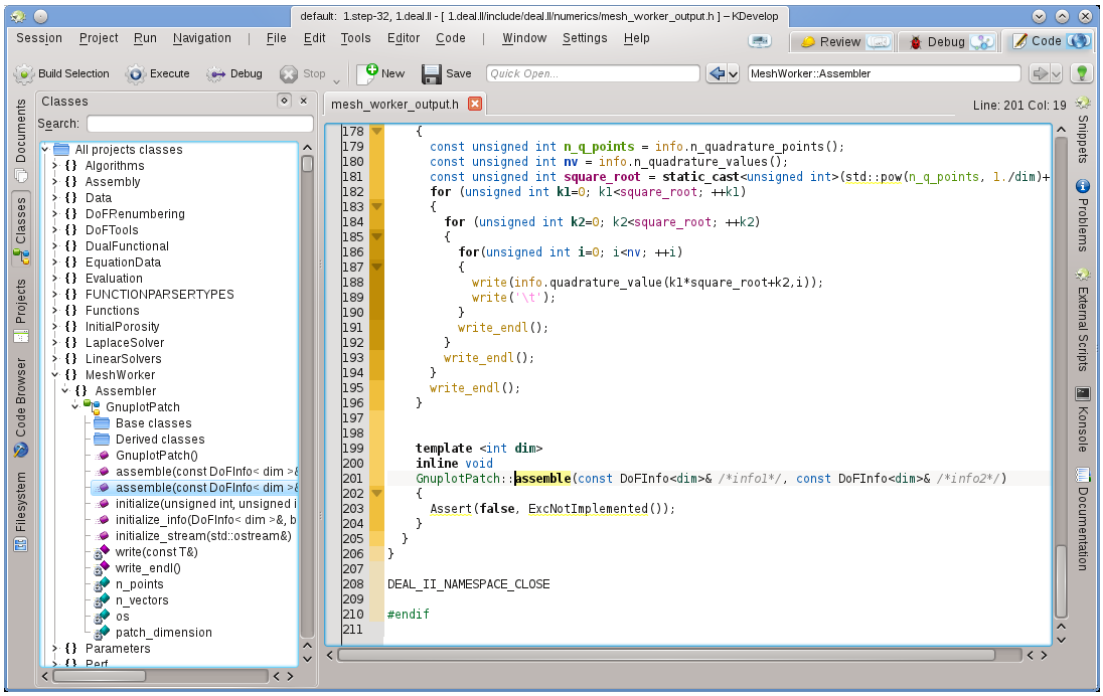


Aqui apresenta uma lista dos espaços de nomes, classes e funções declaradas ou definidas no ficheiro actual, dando-lhe uma visão geral sobre o que se passa neste ficheiro, bem como uma forma de saltar directamente para qualquer uma destas declarações ou definições sem ter que percorrer o ficheiro para cima ou para baixo à procura de um determinado símbolo.

> NOTA
> A informação apresentada a nível do ficheiro é a mesma apresentada no modo de 'Contorno' da navegação do código-fonte; a diferença é que o modo de contorno é apenas uma dica temporária.

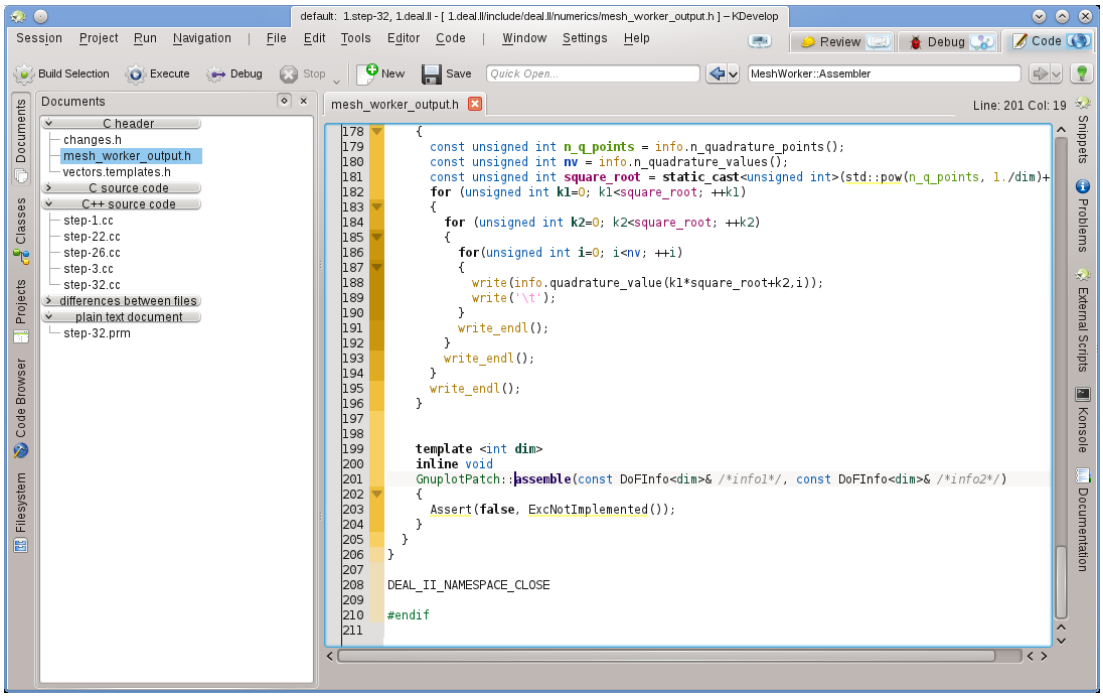
3.2.3 Informação ao nível do projecto e da sessão

Existem muitas formas de obter informações acerca de um projecto inteiro (ou, de facto, sobre todos os projectos de uma sessão). Este tipo de informação é normalmente indicado através de várias áreas de ferramentas. Por exemplo, a ferramenta de **Classes** oferece uma estrutura em árvore de todas as classes e espaços de nomes envolventes para todos os projectos de uma sessão, em conjunto comas funções-membro e variáveis de cada uma destas classes:



Se passar o rato sobre um item irá obter, mais uma vez, informações sobre o símbolo, a localização da sua declaração e definição e as suas utilizações. Se fizer duplo-click sobre um item desta árvore, irá abrir uma janela do editor na posição em que o símbolo está declarado ou definido.

Mas existem outras formas de olhar para a informação global. Por exemplo, a ferramenta de **Documentos** oferece uma vista sobre um projecto com base nos tipos de ficheiros ou outros documentos que compõem este projecto:



3.2.4 O realce do arco-íris explicado

O KDevelop usa uma variedade de cores para realçar os diferentes objectos no código-fonte. Se souber o que as diferentes cores significam, poderá extrair muito rapidamente bastantes informações a partir do código-fonte, bastando para tal olhar para as cores, sem ter de ler um único carácter. As regras de realce são as seguintes:

- Os objectos do tipo Classe / Estrutura, Enumerado (os valores e o tipo), as funções (globais) e os membros das classes têm cada um a sua cor atribuída (as classes são verdes, os enumerados são vermelhos escuros e os membros são amarelos escuros ou violetas, sendo que as funções globais são sempre violetas).
- Todas as variáveis globais aparecem a verde escuro.
- Os identificadores de 'typedefs' de cada tipo aparecem a castanho.
- Todas as declarações e definições de objectos aparecem a negrito.
- Se um membro for acedido dentro do contexto da sua definição (classe de base ou derivada), aparece a amarelo, caso contrário aparece a violeta.
- Se um membro for privado ou protegido, aparece numa cor ligeiramente mais escura quando for usado.
- Para as variáveis locais de um dado bloco de código, as cores do arco-íris são escolhidas com base num código do identificador. Este inclui os parâmetros dessa função. Um identificador terá sempre a mesma cor dentro do seu âmbito (embora o mesmo identificador possa obter uma cor diferente se representar um objecto diferente, isto é se for definido de novo noutro nível), pelo que irá obter a mesma cor para o mesmo identificador em âmbitos diferentes. Como tal, se tiver várias funções que recebam parâmetros com os mesmos nomes, os argumentos irão ficar com cores iguais. Estas cores do arco-íris poderão ser desactivadas em separado da coloração global da janela de configuração.
- Os identificadores para os quais o KDevelop não pôde determinar a declaração correspondente aparecem a branco. Isto poderá acontecer algumas vezes por instruções `#include` em falta.
- Para além dessa coloração, o realce de sintaxe normal do editor será aplicado, como acontece no Kate. O realce semântico do KDevelop irá sempre substituir o realce de sintaxe do editor, caso exista algum conflito.

3.3 Navegar pelo código-fonte

Na secção anterior, discutimos a exploração do código-fonte, isto é obter informações sobre os símbolos, ficheiros e projectos. O passo seguinte é então navegar pelo mesmo, isto é circular por ele todo. Existem de novo vários níveis possíveis para tal: local, dentro de um ficheiro ou dentro de um projecto.

> NOTA
> Muitas das formas de navegar pelo código estão acessíveis através do menu **Navegar** da janela principal do KDevelop.

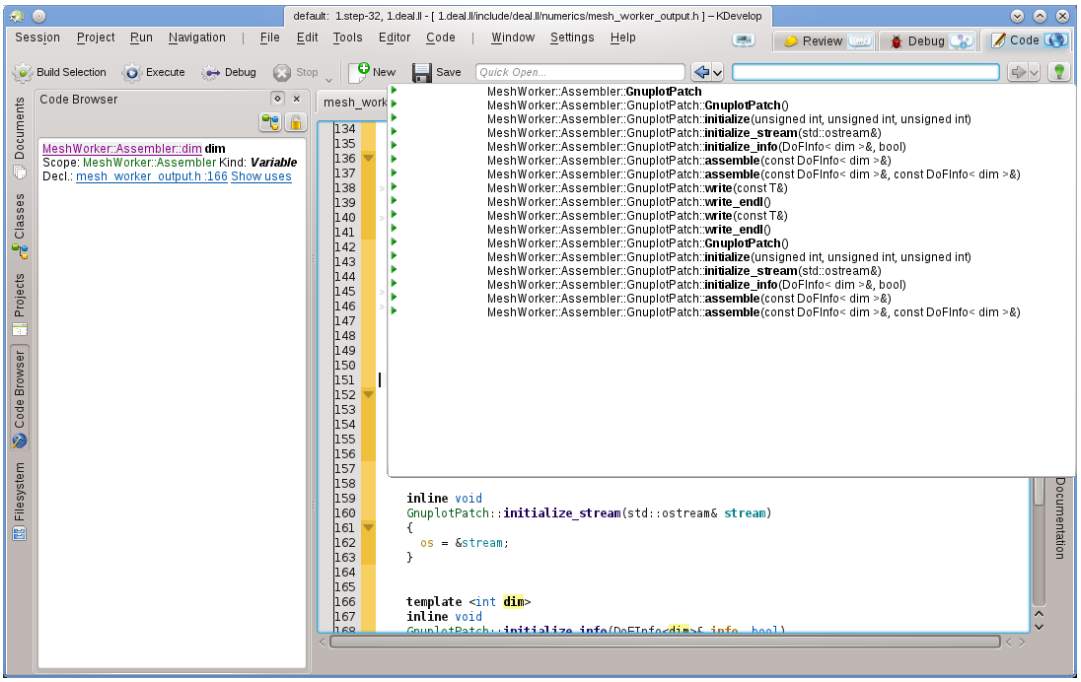
3.3.1 Navegação local

O KDevelop é muito mais que um editor, mas *também* é um editor de código. Como tal, obviamente poderá mover o cursor para cima, baixo, esquerda ou direita num ficheiro de código. Poderá também usar as teclas **PageUp** e **PageDown**, assim como todos os comandos a que está habituado em qualquer outro editor útil.

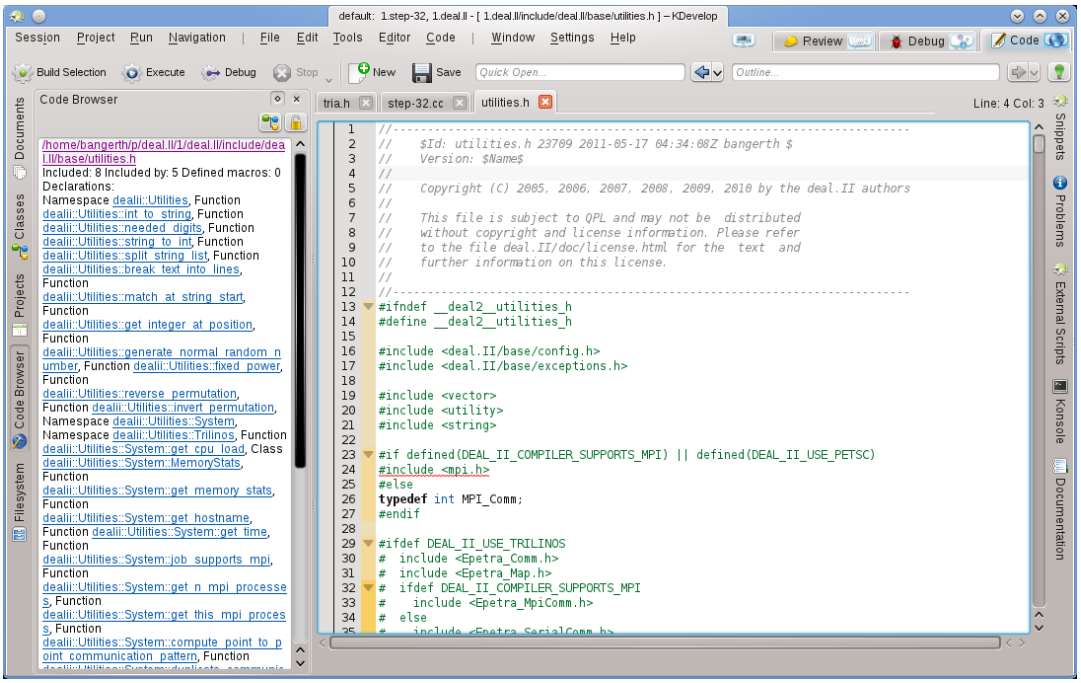
3.3.2 Navegação ao nível do ficheiro e modo de contorno

Ao nível do ficheiro, o KDevelop oferece muitas formas possíveis de navegar pelo código-fonte. Por exemplo:

- **Contorno**: Poderá ter uma visão geral do que se encontra no ficheiro actual, pelo menos de três formas diferentes:
  - Se carregar na área de **Contorno** no canto superior direito da janela principal, ou se carregar em **Alt-Ctrl-N**, irá abrir uma lista que apresenta todas as declarações de funções e classes:



    Poderá então seleccionar para onde desejar saltar ou — se existirem bastantes — começar a escrever o texto que possa aparecer nos nomes apresentados; nesse caso, à medida que vai escrevendo, a lista vai ficando cada vez menor, dado que os nomes não correspondentes ao texto introduzido por si vão sendo retirados, até que esteja pronto para seleccionar uma das opções.
  - Posicionando o cursor ao nível do ficheiro (isto é fora de qualquer declaração ou definição de funções ou classes) e tendo a ferramenta do **Navegador do Código** aberta:



Isto também lhe dá uma ideia geral do que se passa no ficheiro actual, permitindo-lhe seleccionar para onde deseja ir.

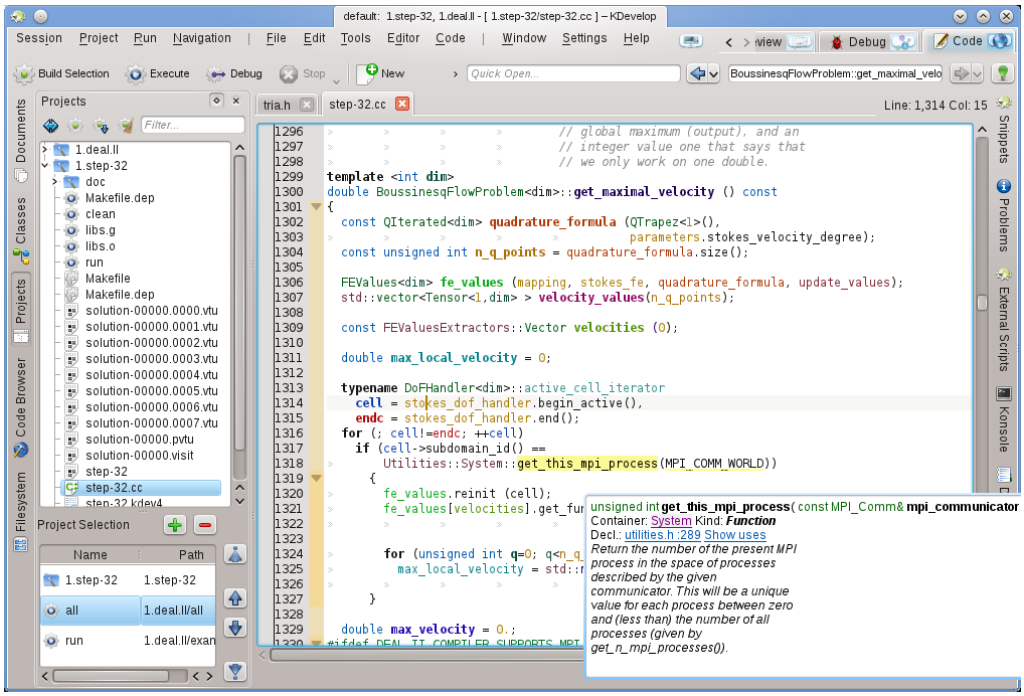
  - Se passar o rato sobre o separador da página de um dos ficheiros abertos também lhe dará uma visão geral do ficheiro nessa página.

- Os ficheiros de código estão organizados como uma lista de declarações ou definições de funções. Se carregar em **Alt-Ctrl-PgUp** e **Alt-Ctrl-PgDown**, irá respectivamente para a definição de função anterior ou seguinte neste ficheiro.

3.3.3 Navegação ao nível do projecto e sessão. Navegação semântica

Como foi mencionado noutros locais, o KDevelop não tem em consideração normalmente os ficheiros de código individuais mas olha sim para os projectos como um todo (ou para todos os projectos que façam parte da sessão actual). Por consequência, oferece várias possibilidades para navegar pelos projectos inteiros. Alguns destes são derivados do que já foi discutido na secção como Explorar o código-fonte, enquanto outras são completamente diferentes. O tema em comum é que estas funcionalidades de navegação baseiam-se numa *compreensão semântica* do código, isto é elas oferecem-lhe algo que necessitar de processar os projectos por inteiro e interligar os dados. A seguinte lista mostra-lhe algumas formas de navegar pelo código-fonte que esteja espalhado por uma grande quantidade de ficheiros:

- Como foi visto na secção sobre Explorar o código-fonte, poderá obter uma dica que explica os nomes dos espaços de nomes, classes, funções ou variáveis individuais, passando o cursor do seu rato sobre eles ou mantendo a tecla **Alt** carregada durante algum tempo. Aqui está um exemplo:



Se carregar nas ligações para a declaração de um símbolo ou se expandir a lista de utilizações, poderá saltar para esses locais, abrindo se necessário o respectivo ficheiro e colocando o cursor na posição correspondente. Poderá obter um efeito semelhante se usar a ferramenta do **Navegador de Código**, que também foi descrita anteriormente.

- **Abertura rápida**: Uma forma bastante poderosa de saltar para outros ficheiros ou locais é usar os vários métodos de *abertura rápida* no KDevelop. Existem quatro versões destes:
  - **Abrir rapidamente a classe** (**Navegar** → **Abrir rapidamente a classe** ou **Alt-Ctrl-C**): Irá obter uma lista com todas as classes nesta sessão. Comece a escrever (uma parte de) o nome de uma classe para que a lista se vá reduzindo para mostrar apenas as que corresponderem ao texto escrito por si até agora. Se a lista for pequena o suficiente, seleccione um elemento, com as teclas de cursor para cima ou baixo, para que o KDevelop o leve para o local em que a classe está declarada.
  - **Abrir rapidamente a função** (**Navegar** → **Abrir rapidamente a função** ou **Alt-Ctrl-M**): Irá obter uma lista com todas as funções 'membros' que fazem parte dos projectos na sessão actual, podendo seleccionar, a partir desta, da mesma forma que foi descrito acima. Lembre-se que esta lista poderá incluir tanto as declarações como as definições das funções.
  - **Abrir rapidamente o ficheiro** (**Navegar** → **Abrir rapidamente o ficheiro** ou **Alt-Ctrl-O**): Irá obter uma lista com todos os ficheiros que fazem parte dos projectos na sessão actual, onde poderá escolher o ficheiro em questão da mesma forma que foi descrita acima.
  - **Abertura rápida universal** (**Navegar** → **Abertura rápida** ou **Alt-Ctrl-Q**): Se se esquecer da combinação de teclas associada a algum dos comandos acima, este é o 'canivete suíço' universal — apresenta-lhe simplesmente uma lista combinada com todos os ficheiros, funções, classes e outros itens que possa seleccionar.

- **Ir para a declaração/definição**: Ao implementar uma função-membro, normalmente uma pessoa precisa de voltar ao ponto em que foi declarada uma função, por exemplo para manter a lista de argumentos da função sincronizada entre a declaração e a definição ou para actualizar a documentação. Para o fazer, coloque o cursor sobre o nome da função e seleccione a opção **Navegação** → **Ir para a declaração** (ou carregue em **Ctrl-.**) para ir para o local onde está declarada a função. Existem várias formas de voltar ao local original:
  - Seleccionando a opção **Navegação** → **Ir para a definição** (ou carregando em **Ctrl-,**).

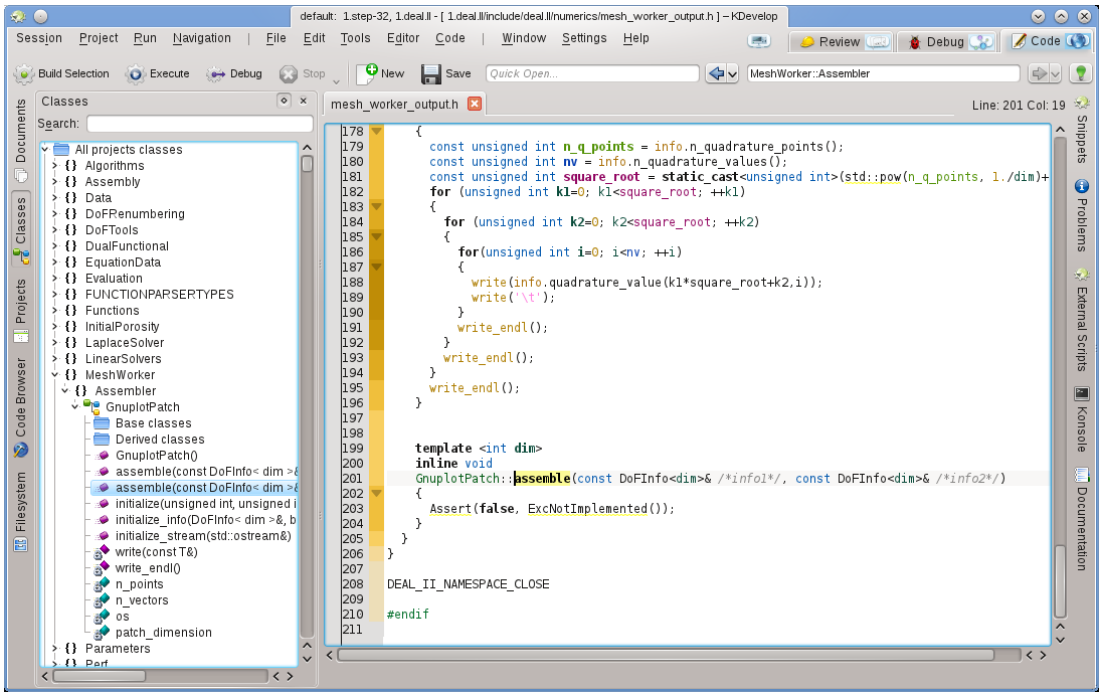
– Seleccionando a opção **Navegação** → **Contexto visitado anterior** (ou carregando em **Meta-Esquerda**), como descrito em baixo.

NOTA
Ir para a declaração de um símbolo é algo que não só funciona quando colocar o cursor sobre o nome da função que se encontra a implementar de momento, mas também funciona para outros símbolos: se colocar o cursor sobre uma variável (local, global ou membro) e for para a sua declaração, irá também levá-lo para a localização da sua declaração. Da mesma forma, poderá colocar o cursor sobre o nome de uma classe, por exemplo sobre a declaração da variável de uma função, e ir para o local da sua declaração.

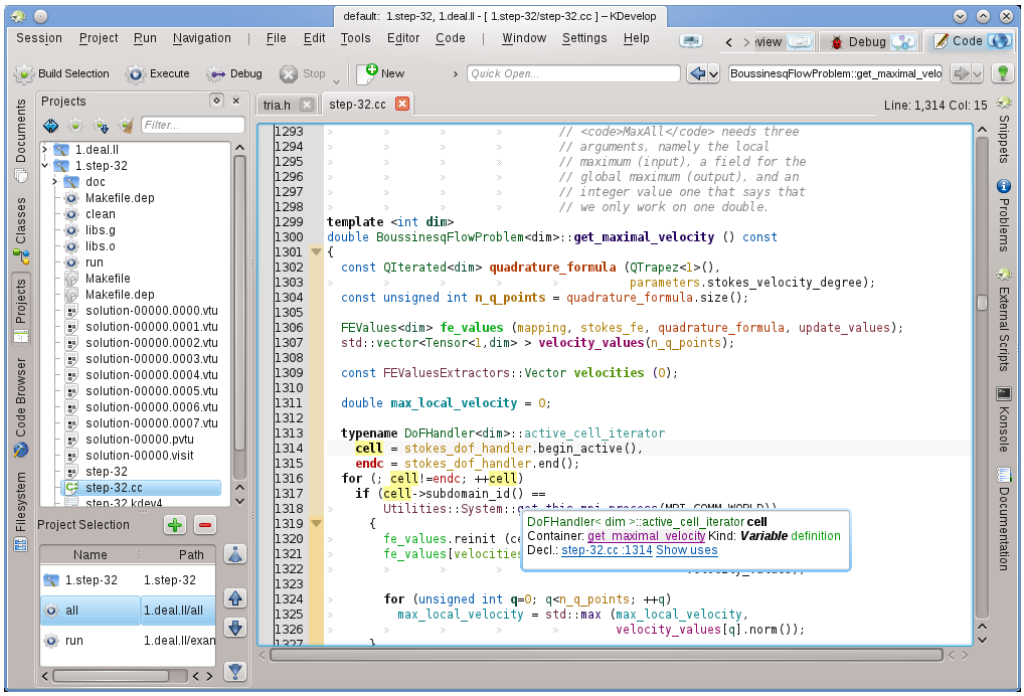
- **Alternar entre a declaração/definição**: No exemplo acima, para ir para o local da declaração da função actual, terá primeiro de colocar o cursor sobre o nome da função. Para evitar este passo, poderá seleccionar a opção **Navegação** → **Alternar entre a definição/declaração** (ou carregue em **Shift-Ctrl-C**) para ir para a declaração da função onde se encontra o cursor de momento. Se seleccionar uma segunda vez a mesma opção, voltará para o local em que está definida a função.
- **Uso anterior/seguinte**: Se colocar o cursor sobre o nome de uma variável local e seleccionar a opção **Navegação** → **Uso seguinte** (ou carregar em **Meta-Shift-Direita**) irá para a utilização seguinte desta variável no código. (Lembre-que isto não pesquisa apenas pela ocorrência seguinte da variável mas também tem em conta variáveis com o mesmo nome, mas em âmbitos diferentes). O mesmo resulta para a utilização dos nomes das funções. Se seleccionar **Navegação** → **Uso anterior** (ou carregar em **Meta-Shift-Esquerda**), irá para a utilização anterior de um dado símbolo.

NOTA
Para ver uma lista com todas as utilizações de um nome, coloque o cursor sobre ele e abra a ferramenta do **Navegador do Código** ou carregue e mantenha carregada a tecla **Alt**. Isto é explicado com mais detalhe na secção sobre como Explorar o código.

- A **lista de contextos**: os navegadores Web têm esta funcionalidade, na qual poderá recuar e avançar pela lista das páginas visitadas mais recentemente. O KDevelop tem o mesmo tipo de funcionalidades, excepto que, em vez de páginas Web, você visita os *contextos*. Um contexto é a localização actual do cursor e o utilizador podê-la-á alterar se navegar para fora dela, usando tudo menos os comandos de cursores — por exemplo, se carregar num local indicado por uma dica, na área de ferramentas do **Navegador de Código**, uma das opções indicadas no menu de **Navegação** ou qualquer outro comando de navegação. Se usar as opções **Navegação** → **Contexto Visitado Anterior** (**Meta-Esquerda**) e **Navegação** → **Contexto Visitado Seguinte** (**Meta-Direita**) irá percorrer esta lista de contextos visitados, assim como acontece nos botões para **recuar** e **avançar** num navegador para as páginas Web visitadas.
- Finalmente, existem áreas de ferramentas que lhe permitem navegar para diferentes locais do seu código. Por exemplo, a ferramenta de **Classes** oferece-lhe uma lista com todos os espaços de nomes e classes de todos os projectos da sessão actual, permitindo-lhe expandi-la para ver as funções e variáveis membros de cada uma destas classes:



Se fizer duplo-click sobre um item (ou se percorrer o menu de contexto com o botão direito do rato) poderá ir para a localização de declaração do item. Outras ferramentas permitem coisas do género; por exemplo, a área de **Projectos** oferece uma lista dos ficheiros que fazem parte de uma sessão:



Mais uma vez, se fizer duplo-click sobre um ficheiro, abri-lo-á.

3.4 Escrever código-fonte

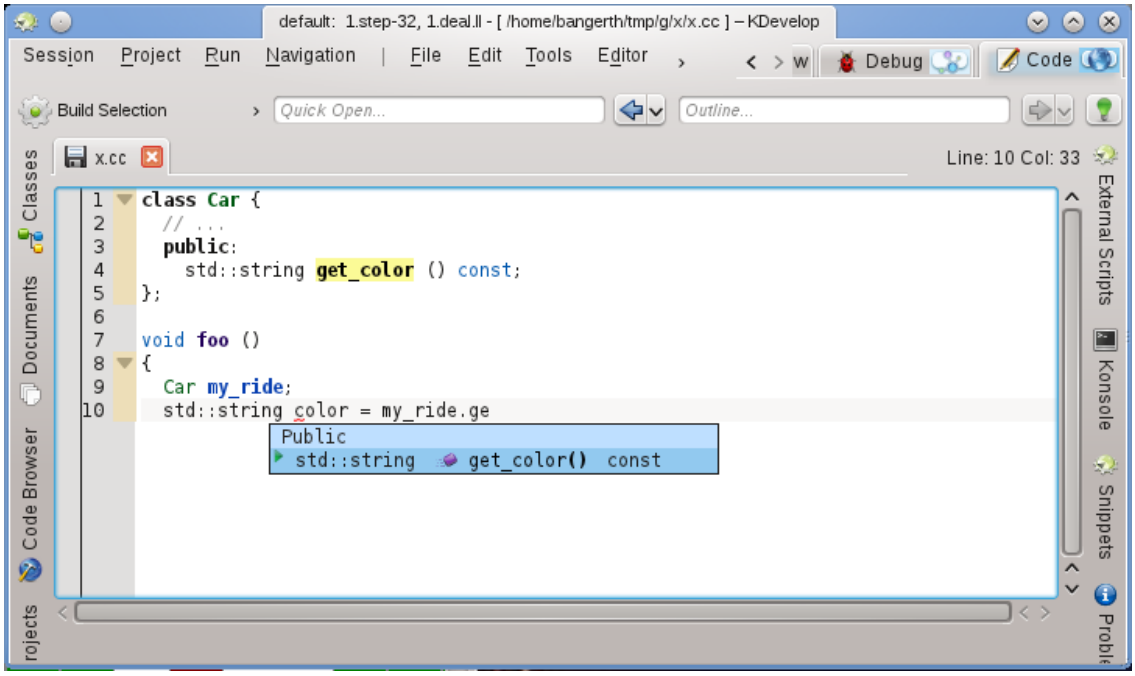
Dado que o KDevelop compreende o código-fonte dos seus projectos, podê-lo-á ajudar a escrever mais código. Os pontos seguintes descrevem algumas das formas como ele o poderá fazer.

3.4.1 Completação automática

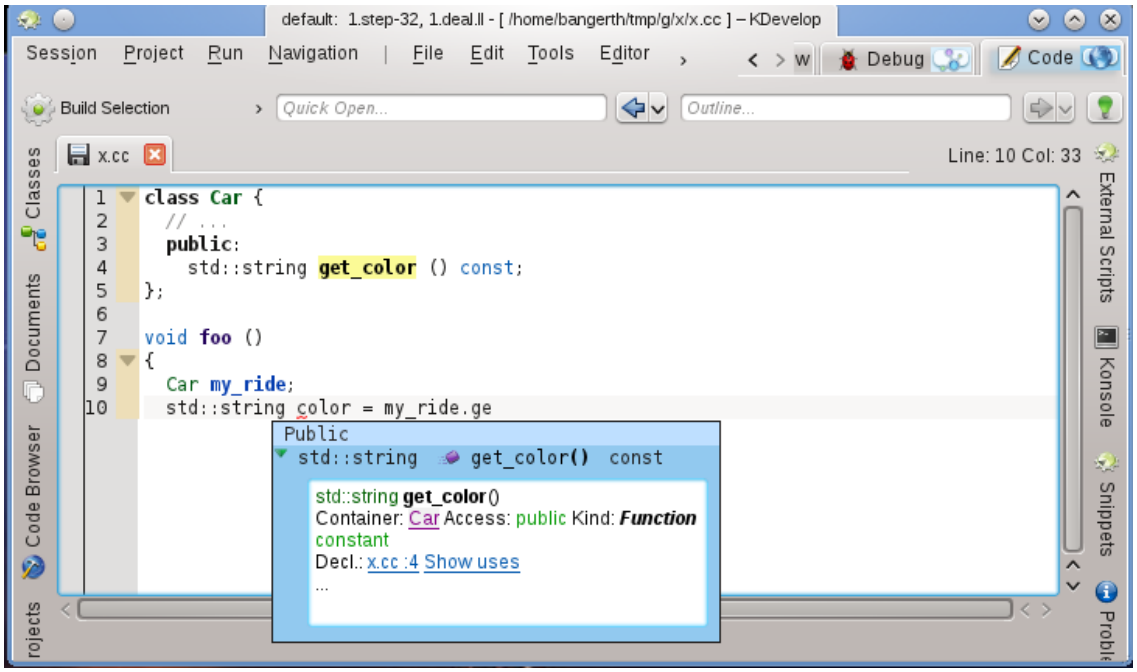
Provavelmente a funcionalidade mais útil de todas na escrita de código novo é a completação automática. Considere, por exemplo, o seguinte pedaço de código:

```
class Carro {
  // ...
  public:
    std::string cor () const;
};
void xpto()
{
  Carro o_meu_carro;
  // ...fazer algo com esta variável...
  std::string cor = o_meu_carro.co
```

Na última linha, o KDevelop irá recordar que a variável `o_meu_carro` é do tipo `Carro`, como tal, irá oferecer-se para terminar o nome da função-membro `co` como `cor`. De facto, tudo o que tem a fazer é continuar a escrever até que a funcionalidade de completação automática tenha reduzido o número de ocorrências a uma, carregando então na tecla **Enter**:



Lembre-se que poderá carregar sobre a dica para obter mais informações acerca da função, para além do seu tipo devolvido e se é pública ou não:



A completação automática poder-lhe-á poupar bastante escrita se o seu projecto usar nomes de variáveis e funções compridos; para além disso, evita os enganos nos nomes (e os erros de compilação daí resultantes) e torna muito mais simples recordar os nomes exactos das funções; por exemplo, se todos os seus métodos de leitura começarem por `get_` (ler_), então a funcionalidade de completação automática poderá apresentar uma lista com todos os métodos de leitura possíveis, logo que tenha escrito as primeiras quatro letras, recordando-o possivelmente no processo qual a função correcta. Lembre-se que, para a completação automática funcionar, nem a declaração da classe `Carro` nem da variável `o_meu_carro` terão de estar no mesmo ficheiro onde está a escrever o código de momento. O KDevelop simplesmente tem de saber onde estão ligadas estas classes e variáveis, isto é os ficheiros aos quais é necessário ter estas ligações feitas terão de fazer parte do projecto onde está a trabalhar.

NOTA

O KDevelop nem sempre sabe quando o deverá assistir a completar o código. Se a dica de completação automática não abrir automaticamente, carregue em **Ctrl-Espaço** para abrir uma lista de completações manualmente. De um modo geral, para a completação automática funcionar, o KDevelop precisa de processar os seus ficheiros de código. Isto acontece em segundo plano para todos os ficheiros que fizerem parte dos projectos da sessão actual, após iniciar o KDevelop, assim como após o utilizador terminar de escrever durante uma fracção de segundo (o atraso pode ser configurado).

NOTA

O KDevelop só processa ficheiros que ele considere como sendo código-fonte, de acordo com o tipo MIME do ficheiro. Este tipo não está definido até à primeira vez em que um ficheiro é gravado; por consequência, ao criar um ficheiro novo e ao começar a escrever código, não activará o processamento da completação automática até que seja gravado pela primeira vez.

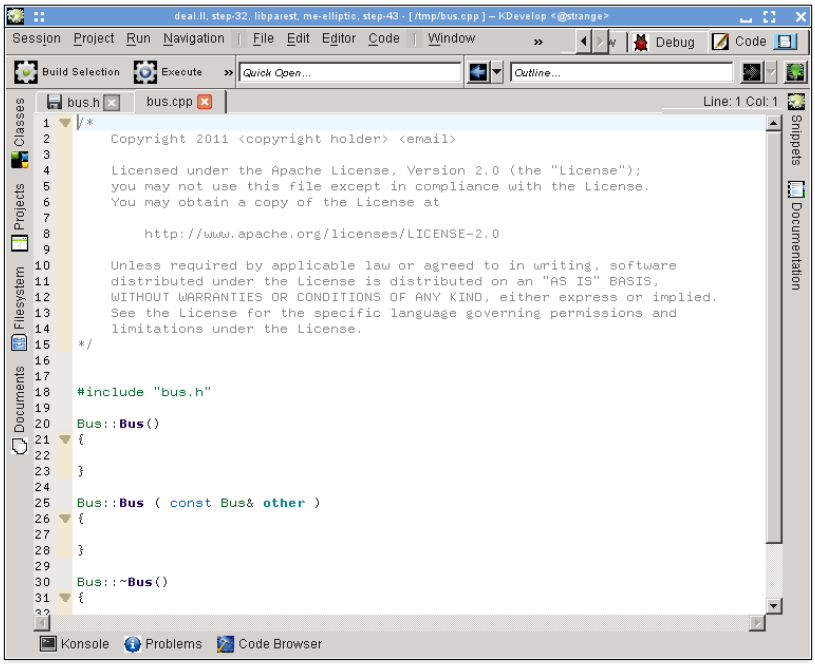
NOTA
Como na nota anterior, para a completação automática funcionar, o KDevelop terá de conseguir descobrir as declarações nos ficheiros de inclusão. Para tal, ele procura num conjunto de locais predefinidos. Se não encontrar automaticamente um ficheiro de inclusão, irá sublinhar o nome de um ficheiro a vermelho; nesse caso, carregue com o botão direito do rato sobre ele para indicar explicitamente ao KDevelop onde é que se encontram estes ficheiros, bem como a informação que fornecem.

NOTA
A configuração da completação automática é discutida nesta secção deste manual.

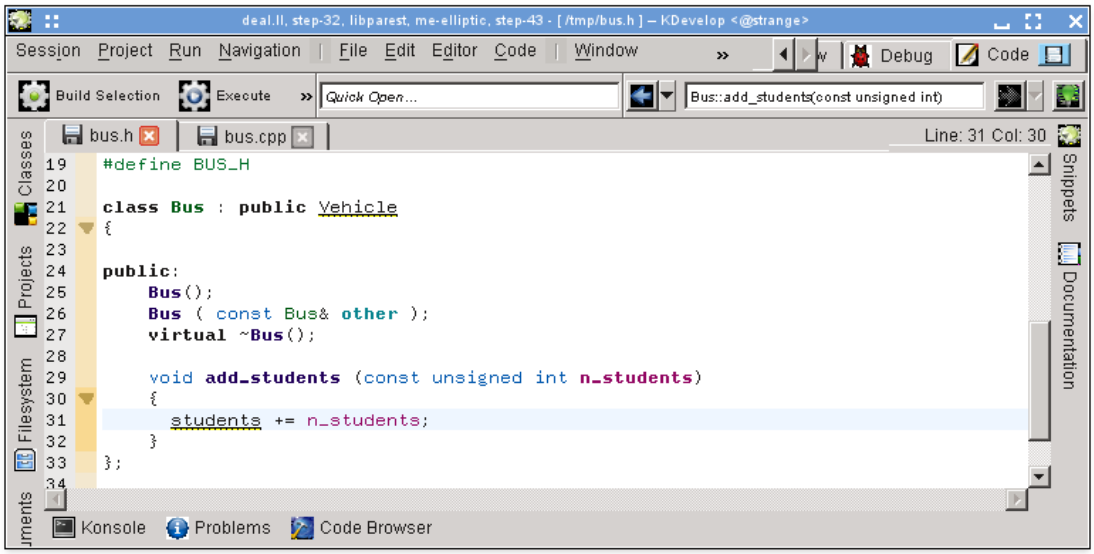
3.4.2 Adicionar classes novas e implementar as funções-membro

O KDevelop tem um assistente para adicionar classes novas. O procedimento está descrito em Criar uma nova classe. Pode criar uma classe simples em C++ com o modelo de C++ Básico, na categoria `Classe`. No assistente, poderá escolher algumas funções predefinidas, como um construtor vazio, um construtor por cópia e um destrutor.

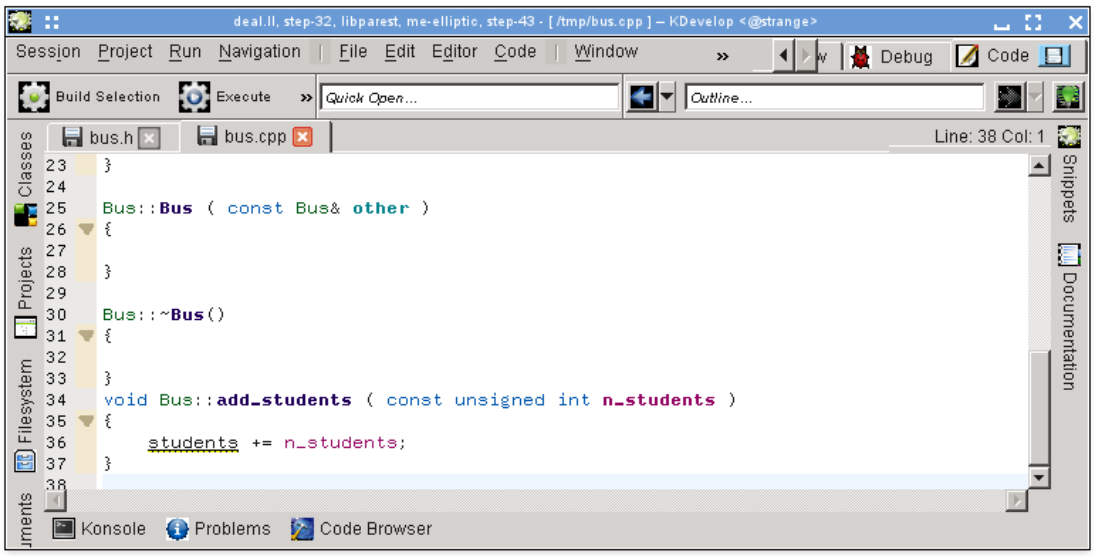
Depois de terminar o assistente, os ficheiros novos estão criados e abertos no editor. O ficheiro de inclusão já contém guardas de inclusão e a classe nova tem todas as funções-membro que seleccionámos. Os dois próximos passos seriam a documentação da classe e das suas funções-membro e a sua respectiva implementação. Iremos discutir algumas ajudas sobre a documentação das classes e funções depois. Para implementar as funções especiais já adicionadas, basta ir para a página **bus.cpp** onde se encontra já o esqueleto das funções:



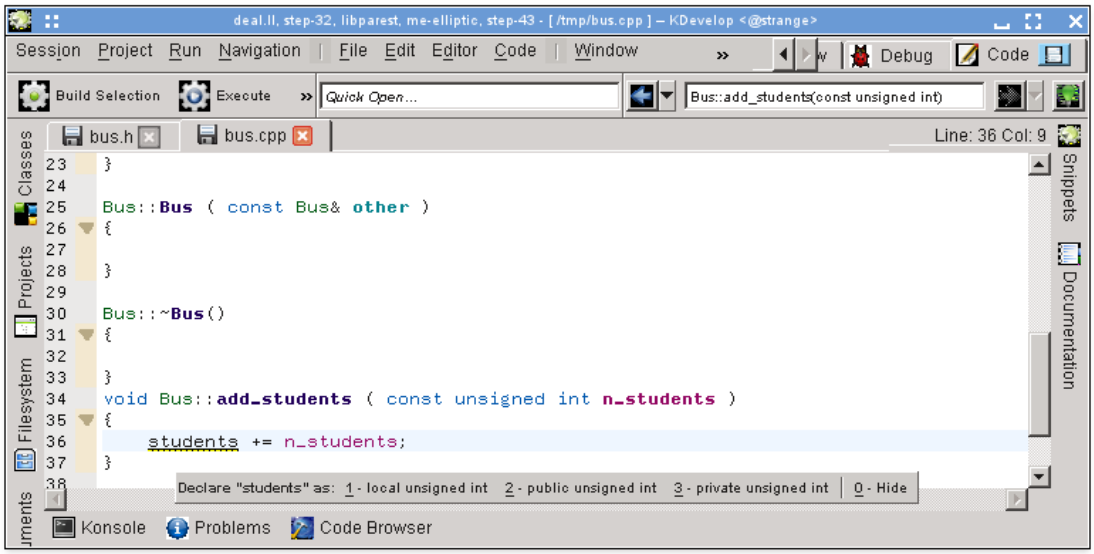
Para adicionar novas funções-membro, volte ao ficheiro **autocarro.h** e adicione o nome de uma função. Por exemplo, adicione o seguinte:



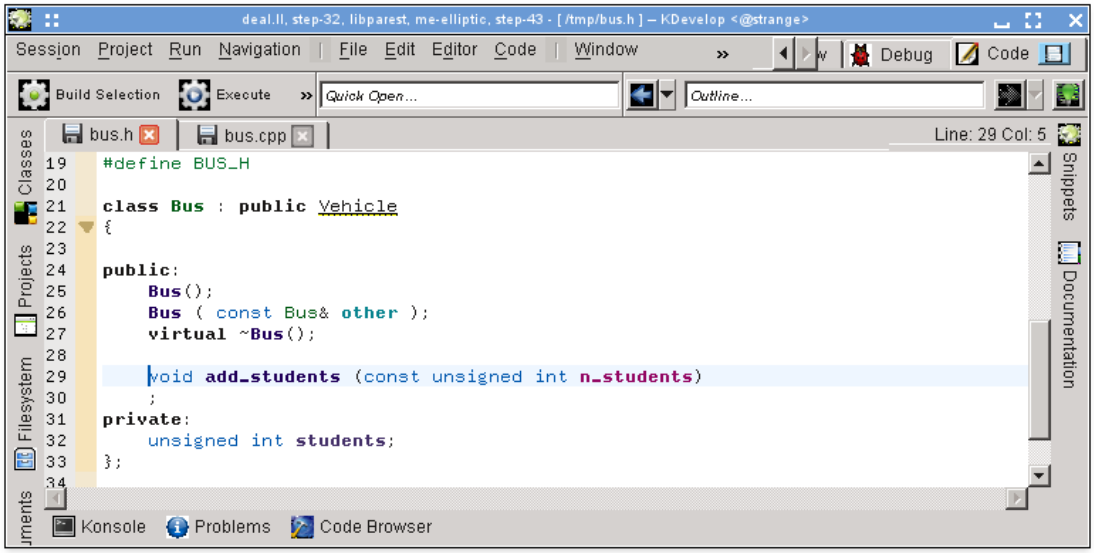
Repare como já foi iniciada a implementação. Contudo, em muitos estilos de código, a função não deveria ser implementada no ficheiro de inclusão mas sim no ficheiro '.cpp' correspondente. Para tal, coloque o cursor sobre o nome da função e seleccione **Código** → **Mover para o código** ou carregue em **Ctrl-Alt-S**. Isto remove o código entre chavetas do ficheiro de inclusão (e substitui-o por um ponto-e-vírgula para terminar a declaração da função) e move-o para o ficheiro de código:



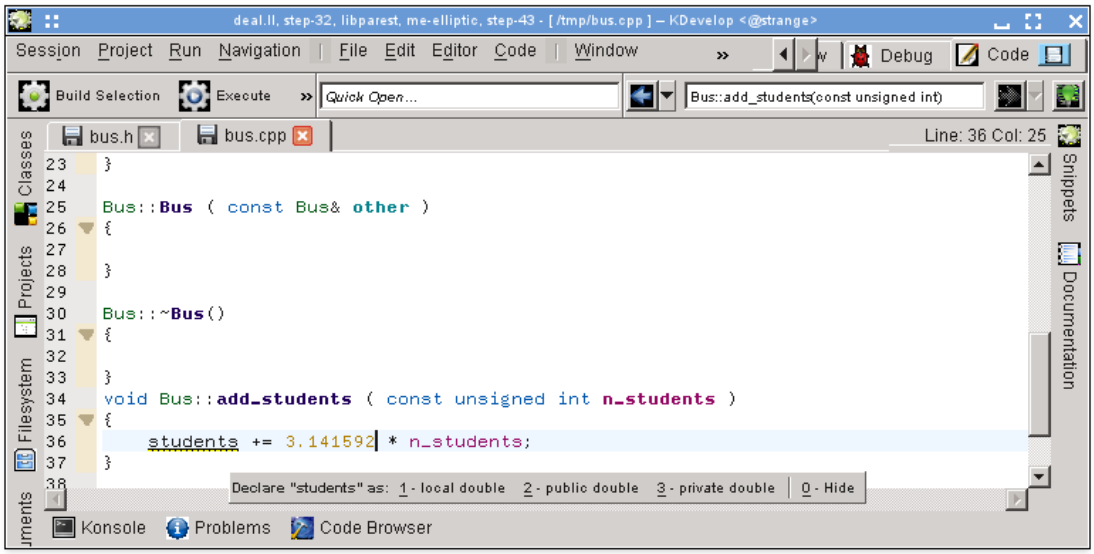
Repare que o autor ainda agora começou a escrever e que desejava inferir que a variável `estudantes` deveria ser provavelmente uma variável-membro da classe `Autocarro`, mas esta ainda não foi adicionada. Repare também como o KDevelop a sublinha para clarificar que ainda não sabe nada sobre a variável. Contudo, este problema pode ser resolvido: se carregar no nome da variável, irá aparecer a seguinte dica:



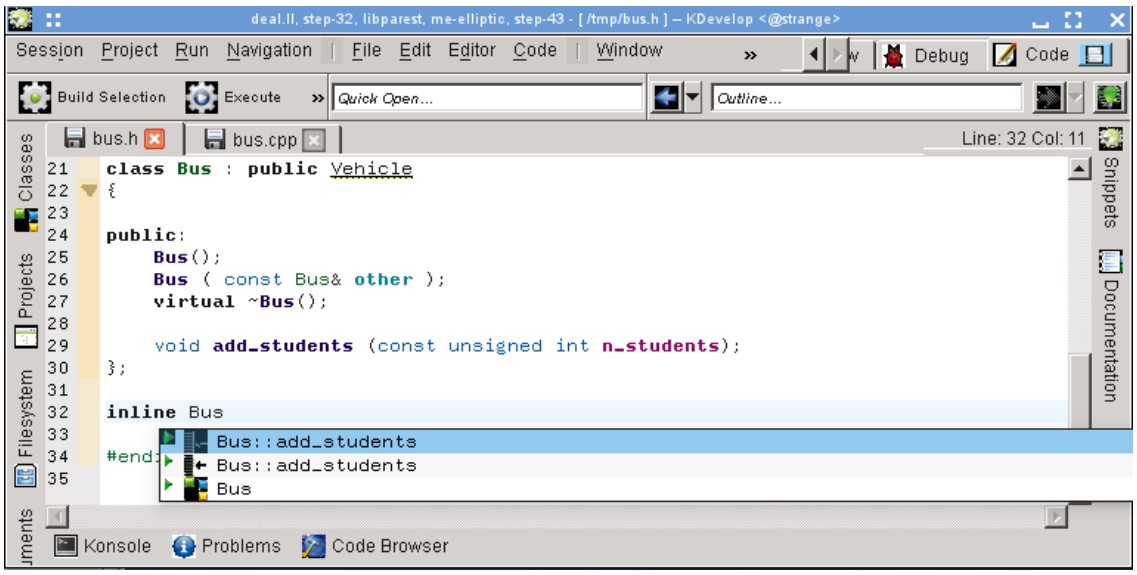
(Conseguirá obter o mesmo se carregar com o botão direito sobre o mesmo e seleccionar **Resolver: Declarar Como**.), podendo seleccionar '3 - private unsigned int' (com o rato, ou carregando em **Alt-3**) e ver como é que irá aparecer no ficheiro de inclusão:



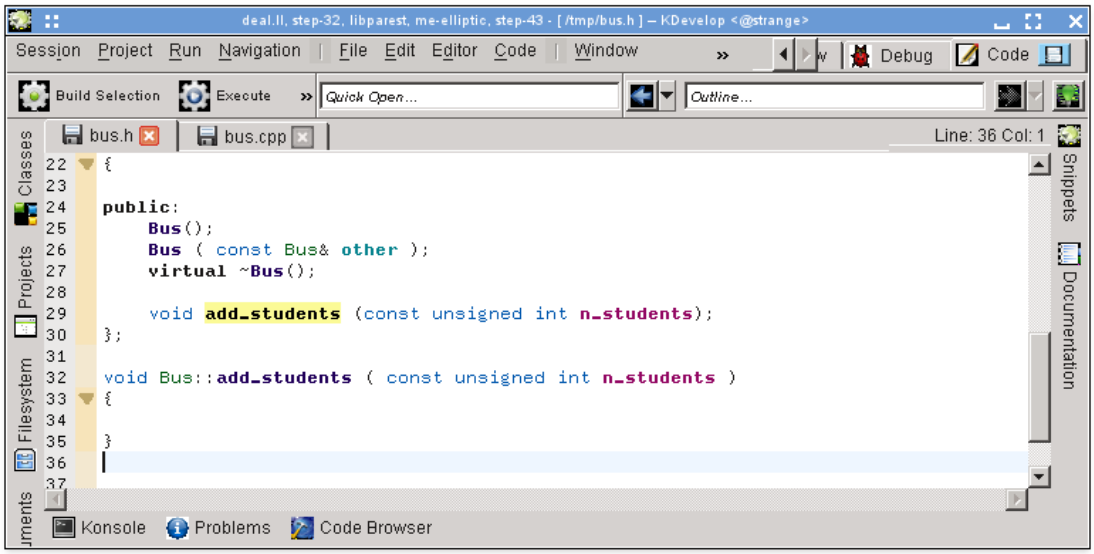
Importa referir que o KDevelop extrai o tipo da variável a declarar a partir da expressão usada para o inicializar. Por exemplo, se tivéssemos escrito a soma na seguinte forma, ainda que dúbia, ele teria sugerido que a variável fosse declarada como `double`:



Como ponto final: O método que usa o **Código** → **Mover para o código** nem sempre insere a nova função-membro onde a deseja. Por exemplo, poderá querer marcá-la como `inline` e colocá-la no fundo do ficheiro de inclusão. Se for esse o caso, escreva a declaração e comece a escrever a definição da função da seguinte forma:



O KDevelop oferece automaticamente todas as completações possíveis do que possa aparecer aqui. Se seleccionar um dos dois `adicionar_estudantes` irá mostrar o seguinte código que já preenche a lista de argumentos completa:




```
{

public:
    Bus();
    Bus ( const Bus& other );
    virtual ~Bus();

    void add_students (const unsigned int n_students);
};

void Bus::add_students ( const unsigned int n_students )
{

}
```

NOTA

No exemplo, ao aceitar uma das escolhas na ferramenta de completação automática, irá mostrar a assinatura correcta, mas infelizmente apaga o marcador `inline` já escrito. Isto foi comunicado como sendo o Erro 274245 do KDevelop.

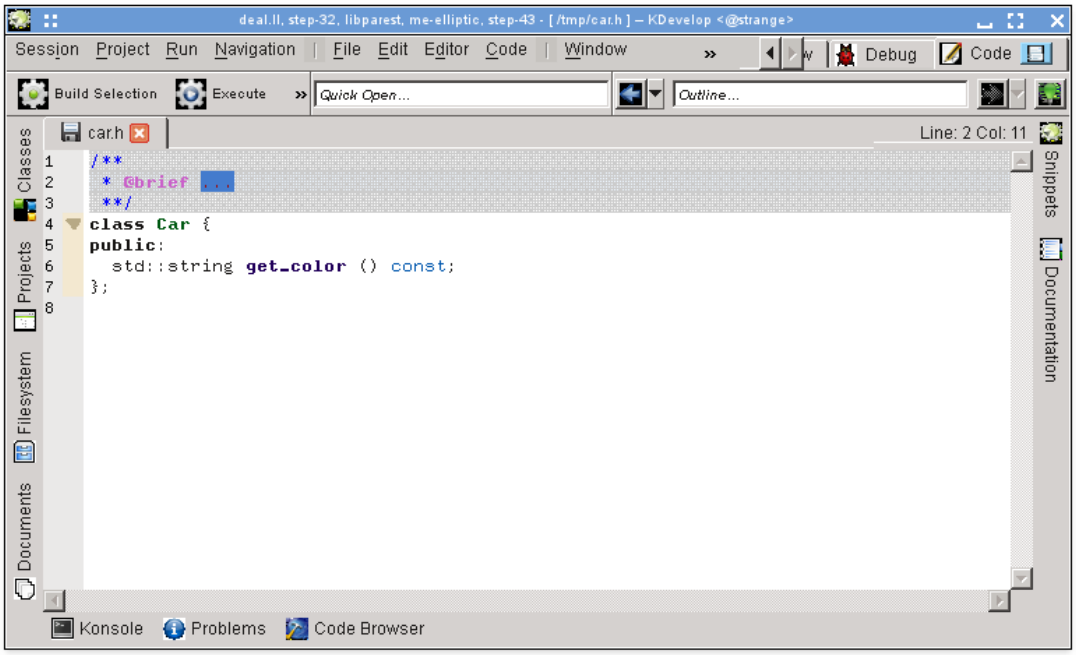
3.4.3 Documentar as declarações

O bom código está bem documentado, tanto ao nível da implementação dos algoritmos dentro das funções, assim como ao nível da interface — isto é, classes, funções (membros e globais) e as variáveis (membros ou globais), com o objectivo de explicar o seu objectivo, os valores possíveis dos argumentos, as pré- e pós-condições, etc. No que diz respeito à documentação da interface, o doxygen tornou-se a norma de facto para formatar os comentários para que possam ser extraídos e apresentados em páginas Web navegáveis.

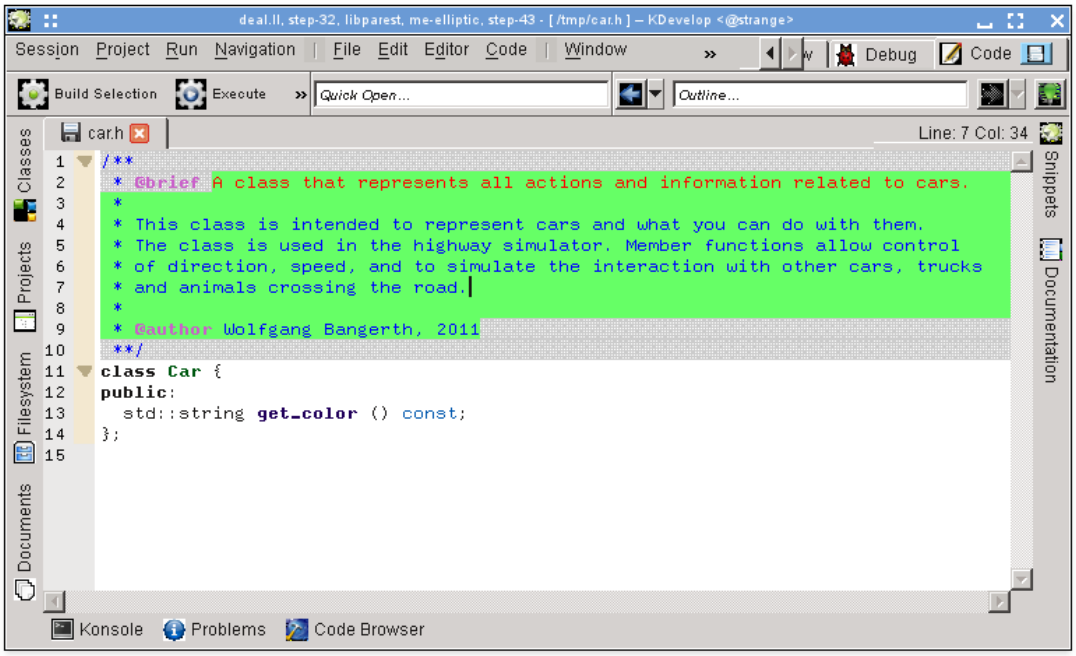
O KDevelop suporta este estilo de comentários, contendo um atalho para gerar a estrutura de comentários que documentam uma classe ou função-membro. Por exemplo, assumindo que já tenha escrito este código:

```
class Carro {
  public:
    std::string cor () const;
};
```

Poderá querer adicionar agora a documentação tanto à classe como à função-membro. Para isso, mova o cursor para a primeira linha e seleccione **Código** → **Documentar a Declaração** ou carregue em **Alt-Shift-D**. O KDevelop irá responder com o seguinte:

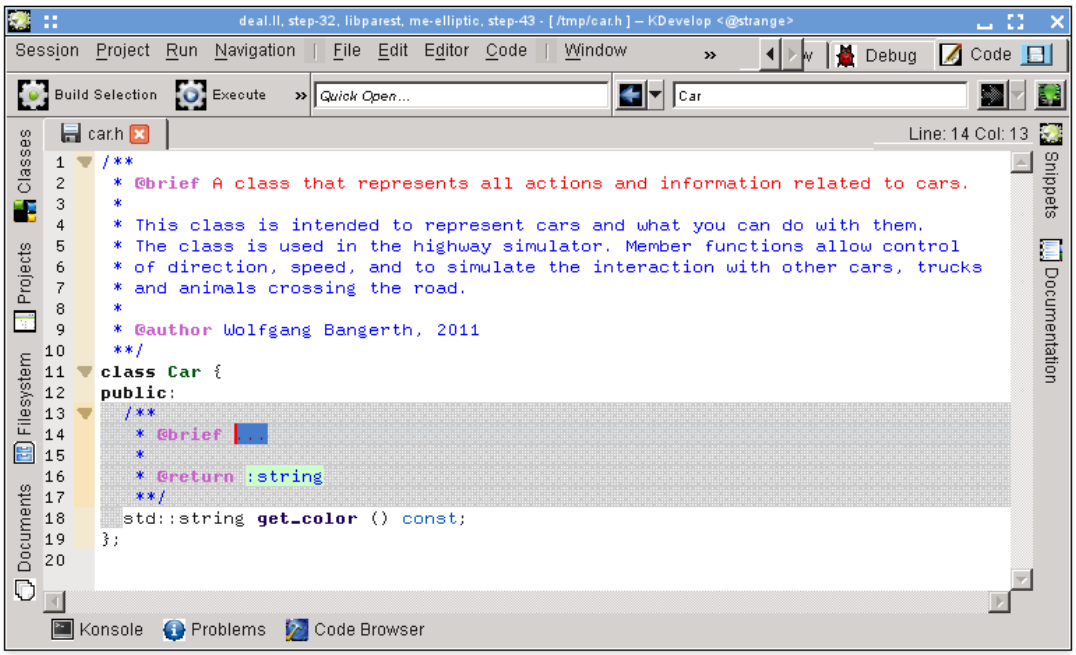


O cursor já se encontra na área a cinzento para você preencher a breve descrição (depois da palavra-chave do 'doxygen' `@brief`) desta classe. Poderá então continuar a adicionar a documentação a este comentário, dando uma cisão mais detalhada sobre o que a classe faz:

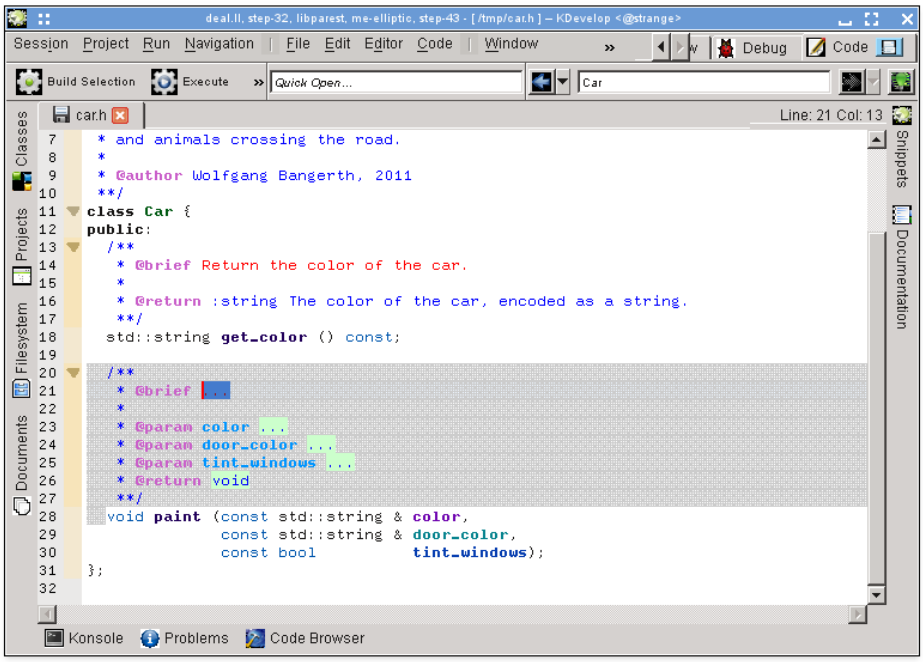


Enquanto o editor estiver dentro do comentário, o texto do mesmo fica realçado a verde (o realce desaparece assim que sair do comentário). Quando for para o fim de uma linha, carregue em **Enter** para que o KDevelop inicie uma nova linha a começar por um asterisco e coloca o cursor com um carácter de indentação.

Agora iremos documentar a função-membro, colocando mais uma vez o cursor sobre a linha da declaração e seleccionando a opção **Código** → **Documentar a Declaração** ou carregando em **Alt-Shift-D**:



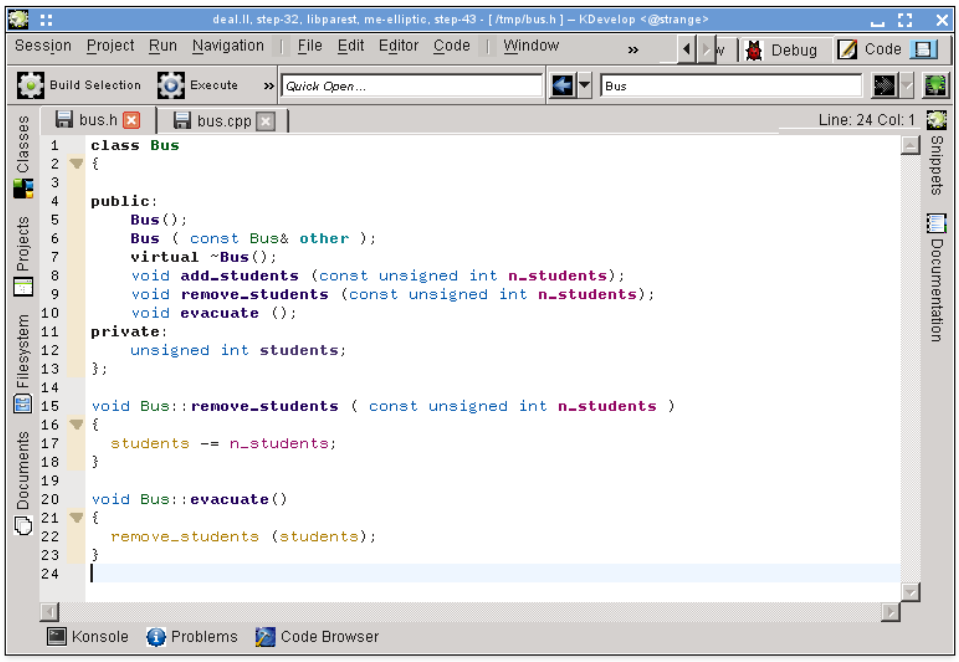
Mais uma vez, o KDevelop irá gerar automaticamente o esqueleto de um comentário, incluindo a documentação da função em si, assim como o tipo devolvido por esta. No caso actual, o nome da função é bastante intuitivo, mas muitas das vezes os argumentos da função poderão não o ser e, como tal, deverão ser documentados individualmente. Para ilustrar isto, vejamos uma função ligeiramente mais interessante e o comentário que o KDevelop irá gerar automaticamente:



Aqui, o comentário sugerido já contém todos os campos do Doxygen dos parâmetros individuais, por exemplo.

3.4.4 Mudar os nomes das variáveis, funções e classes

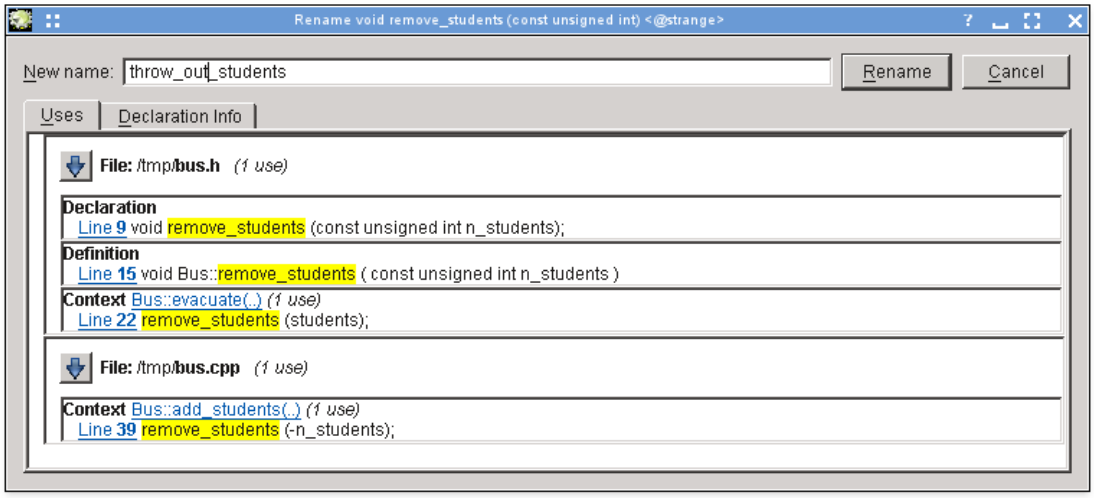
Algumas das vezes, alguém poderá querer mudar o nome de uma função, classe ou variável. Por exemplo, imagine que já tem o seguinte:



Irá então concluir que está insatisfeito com o nome `remover_estudantes` e que se deveria chamar por exemplo `largar_estudantes`. Poderia fazer uma pesquisa-substituição por esse nome, mas isso tem duas desvantagens:

- A função pode ser usada em mais que um ficheiro.
- Realmente só queremos mudar o nome desta função e não tocar nas funções que possam ter o mesmo nome mas que estejam declaradas noutras classes ou espaços de nomes.

Ambos os problemas poderão ser resolvidos se mover o cursor para qualquer uma das ocorrências do nome da função e seleccionar **Código** → **Mudar o nome da declaração** (ou se carregar com o botão direito no nome e seleccionar a opção **Mudar o nome de Auto-carro::remover_estudantes**). Isto irá invocar uma janela onde poderá indicar o novo nome da função e onde poderá ver todos os locais onde esta é usada:

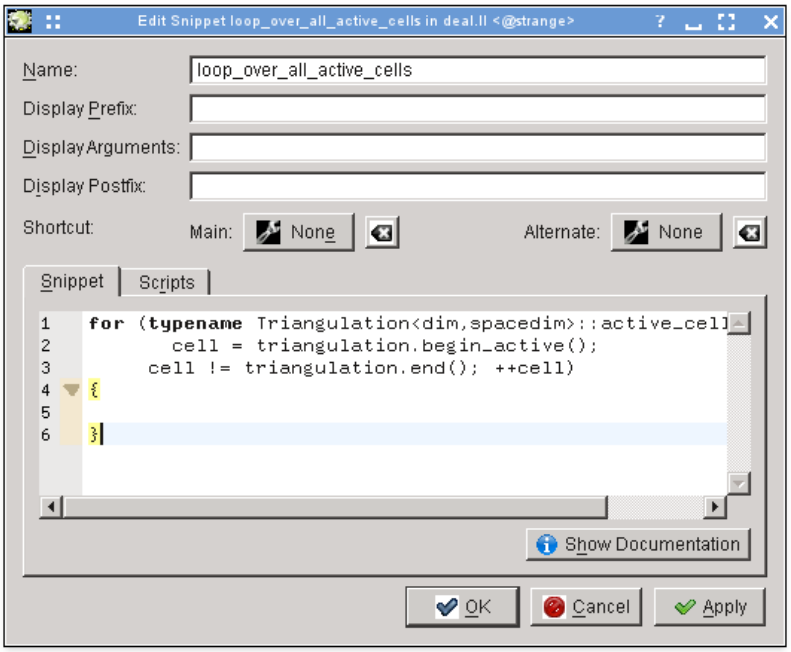


3.4.5 Excertos de código

A maioria dos projectos têm pedaços de código que uma pessoa terá de escrever frequentemente a nível de código-fonte. Os exemplos são: para os criadores de compiladores, um ciclo por todas as instruções; para os criadores de interfaces de utilizador, verificar se os dados do utilizador são válidos e, caso contrário, mostrar uma mensagem de erro; no projecto do autor dessas linhas, o código seria do estilo

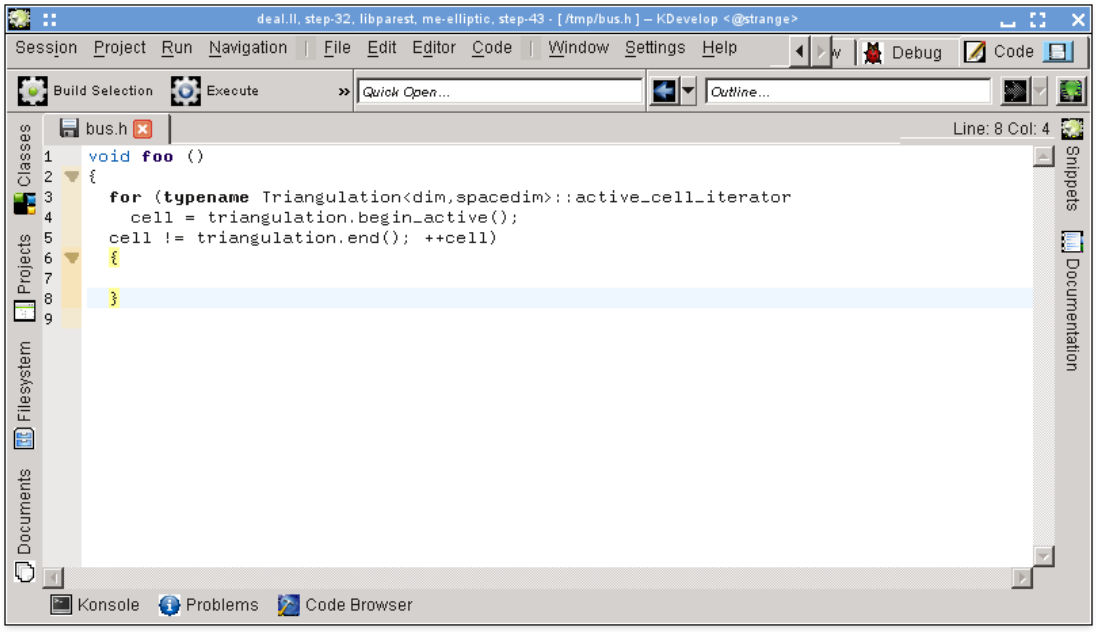
```
for (nometipo Triangulacao::active_cell_iterator
       celula = triangulacao.begin_active();
     celula != triangulacao.end(); ++celula)
  ... fazer algo com a célula ...
```

Em vez de escrever este tipo de texto repetidamente (com todos os erros associados que isso possa introduzir), a ferramenta de **Excertos** do KDevelop podê-lo-á ajudar aqui. Para tal, abra a área de ferramenta (veja em Ferramentas e janelas se o botão correspondente não existir já na envolvência da sua janela). Depois carregue no botão 'Adicionar um repositório' (um nome ligeiramente confuso — ele permite-lhe criar uma colecção de excertos com um dado nome para os ficheiros de código de um dado tipo, isto é código em C++) e crie um repositório vazio. Depois, carregue em  para adicionar um excerto, aparecendo-lhe uma janela como a seguinte:



> **NOTA**
> O nome de um excerto não poderá ter espaços ou outros caracteres especiais, porque deverá ser parecida com o nome de uma função ou variável normal (por razões que se tornarão mais claras no parágrafo seguinte).

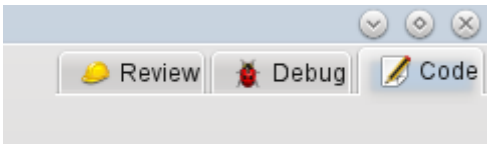
Para usar o excerto assim definido, quando estiver a editar código, basta escrever o nome do excerto como o faria com qualquer função ou variável. Este nome ficará disponível na completação automática — o que significa que não haverá qualquer problema em usar nomes compridos e descritos para um excerto, como o descrito acima — e quando aceitar a dica de sugestão da completação automática (por exemplo, carregando apenas em **Enter**), a parte já introduzida do nome do excerto será substituída pela expansão completa do excerto e será devidamente indentada:



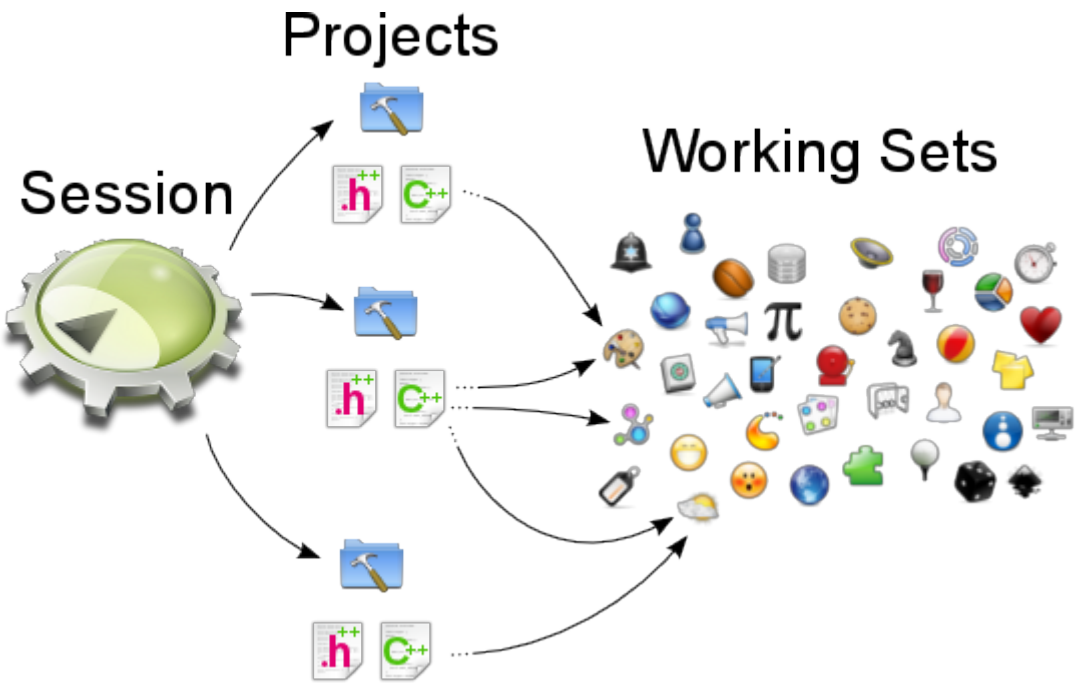
Lembre-se que, para isto funcionar, a ferramenta de **Excertos** não precisa de estar aberta ou visível: só irá precisar da ferramenta para definir excertos novos. Uma alternativa, embora menos conveniente, para expandir um excerto é simplesmente carregar nele na área de ferramentas respectiva.

> NOTA
> Os excertos são muito mais poderosos do que se explicou aqui. Para uma descrição completa do que pode fazer com eles, veja a documentação detalhada sobre a ferramenta de Excertos.

3.5 Modos e conjuntos de trabalho

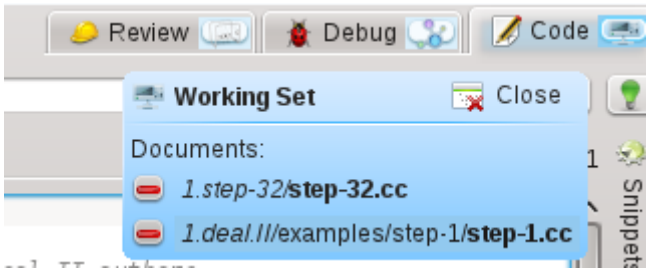


Se tiver chegado até aqui, dê uma vista de olhos na parte superior direita da janela principal do KDevelop. Como aparece na imagem, irá reparar que existem três **modos** possíveis para o KDevelop: **Código** (o modo que discutimos no capítulo anterior ao lidar com o código-fonte), **Depuração** (veja como Depurar os programas) e **Revisão** (veja como Lidar com sistemas de controlo de versões).

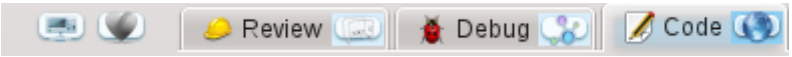


Cada modo tem o seu próprio conjunto de ferramentas espalhadas em torno da janela principal, e cada modo também tem um *conjunto de trabalho* dos ficheiros e documentos abertos de momento. Para além disso, cada um destes conjuntos de trabalho está associado com uma sessão actual, isto é temos a mesma relação apresentada acima. Lembre-se que os ficheiros no conjunto de trabalho vêm da mesma sessão, mas poderão vir de diferentes projectos que façam parte da mesma sessão.

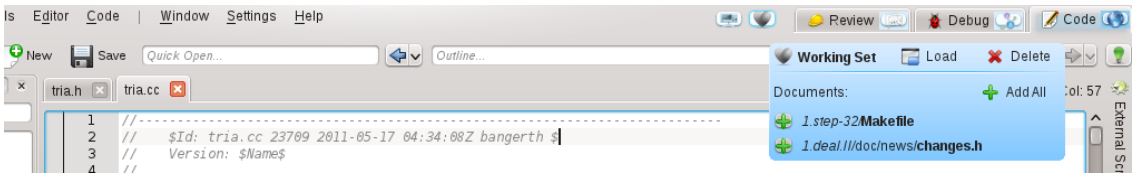
Se abrir o KDevelop da primeira vez, o conjunto de trabalho está vazio — não existem ficheiros abertos. Porém, à medida que abre os ficheiros para os editar (ou depurar ou rever noutros modos quaisquer), o seu conjunto de trabalho vai crescendo. O facto de o seu conjunto de trabalho não estar vazio é indicado através de um símbolo na página, como demonstrado em baixo. Irá reparar que, sempre que fechar o KDevelop e voltar a iniciar de novo, o conjunto de trabalho é gravado e reposto, isto é irá obter o mesmo conjunto de ficheiros abertos.



Se passar o seu rato sobre o símbolo do conjunto de trabalho, irá obter uma dica que lhe mostra os ficheiros que estão abertos de momento neste conjunto de trabalho (aqui: os ficheiros `passo-32.cc` e `passo-1.cc`). Se carregar no sinal de somar vermelho, irá fechar a página do ficheiro correspondente. Talvez ainda mais importante, se carregar no botão com o nome correspondente, irá **fechar** todo o conjunto de trabalho de uma vez (isto é fechar todos os ficheiros abertos de momento). O ponto importante acerca do fecho do conjunto de trabalho, contudo, é que não só fecha todos os ficheiros, como também grava o conjunto e abre um novo, totalmente vazio. Poderá ver isto aqui:



Repare nos dois símbolos à esquerda das páginas dos três modos (o coração e o símbolo não-identificado à sua esquerda). Cada um destes dois símbolos representa um conjunto de trabalho gravado, para além do conjunto aberto de momento. Se passar o seu rato sobre o símbolo do coração, irá obter algo do género:



Isto mostra-lhe que o conjunto de trabalho correspondente contém dois ficheiros e os seus nomes de projectos correspondentes: `Makefile` e `alteracoes.h`. Se carregar em **Carregar**, irá fechar e gravar o conjunto de trabalho actual (que aparece aqui com os ficheiros `tria.h` e `tria.cc` abertos) e irá abrir por seu turno o conjunto seleccionado. Poderá também apagar de forma permanente um conjunto de trabalho, o qual o irá remover da lista de conjuntos de trabalho gravados.

3.6 Algumas combinações de teclas úteis

O editor do KDevelop segue as combinações de teclas padrão para todas as operações de edição normais. Contudo, também suporta um conjunto de operações mais avançado ao editar o código-fonte, estando algumas associadas a combinações de teclas em particular. As seguintes são particularmente úteis:

Circular pelo código	
Ctrl-Alt-O	Abrir rapidamente o ficheiro: indique parte do nome do ficheiro e seleccione entre todos os ficheiros das pastas dos projectos da sessão actual que correspondam ao texto; assim, será aberto o ficheiro
Ctrl-Alt-C	Abrir rapidamente a classe: indique parte do nome de uma classe e seleccione entre todas as classes que corresponderem; o cursor irá então saltar para a declaração da classe
Ctrl-Alt-M	Abrir rapidamente a função: indique parte do nome de uma função (membro) e seleccione entre todos os nomes que corresponderem; repare que a lista mostra tanto as declarações como as definições, pelo que o cursor irá então saltar para o item seleccionado
Ctrl-Alt-Q	Abertura rápida universal: escreva qualquer coisa (nome de um ficheiro, classe ou função) e obtenha uma lista de tudo o que corresponder

Ctrl-Alt-N	Contorno: Oferece uma lista com todas as coisas que estão a acontecer neste ficheiro, isto é declarações de classes e definições das funções
Ctrl-,	Ir para a definição de uma função, caso o cursor esteja de momento sobre a declaração de uma função
Ctrl-.	Ir para a declaração de uma função ou variável, caso o cursor esteja de momento sobre a definição de uma função
Ctrl-Alt-PageDown	Ir para a função seguinte
Ctrl-Alt-PageUp	Ir para a função anterior
Ctrl-G	Ir para a linha

Pesquisa e substituição	
Ctrl-F	Procurar
F3	Procurar o seguinte
Ctrl-R	Substituir
Ctrl-Alt-F	Pesquisa-substituição em vários ficheiros

Outras coisas	
Ctrl-_	Recolher ou fechar um nível: torna este bloco invisível, por exemplo se quiser apenas focar-se na parte macroscópica de uma função
Ctrl-+	Expandir um nível: anula o fecho ou recolhimento
Ctrl-D	Comentar o texto seleccionado ou a linha actual
Ctrl-Shift-D	Retirar o comentário o texto seleccionado ou a linha actual
Alt-Shift-D	Documentar a função actual. Se o cursor estiver sobre a declaração de uma função ou classe, então, ao invocar esta combinação, irá criar um comentário do estilo do 'doxygen' devidamente preenchido com uma listagem de todos os parâmetros, valores devolvidos, etc.
Ctrl-T	Trocar o carácter actual com o anterior
Ctrl-K	Apaga a linha actual (nota: esta não é a opção do 'emacs' para 'apagar daqui até ao fim da linha')

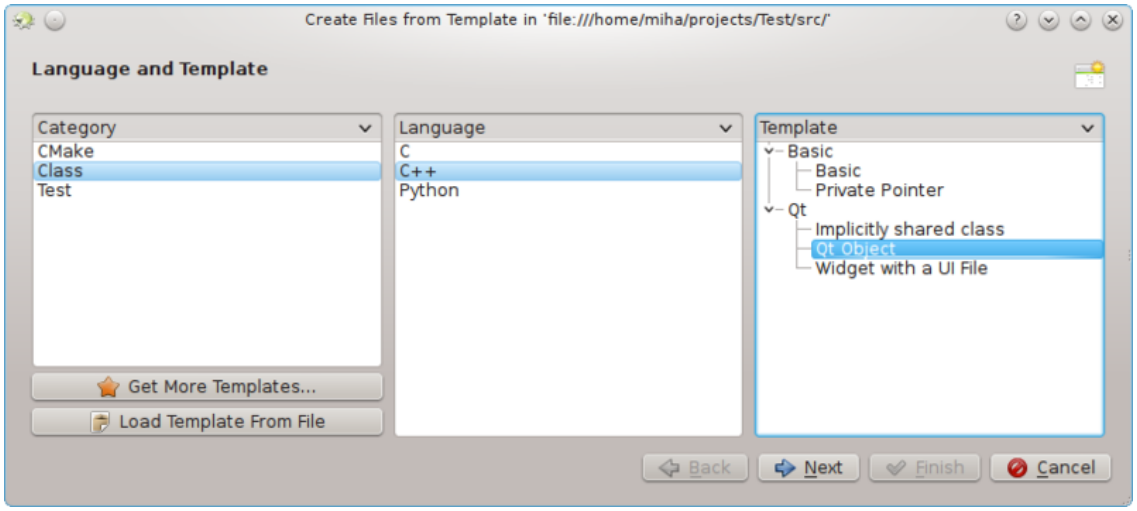
Capítulo 4

Geração de código com modelos

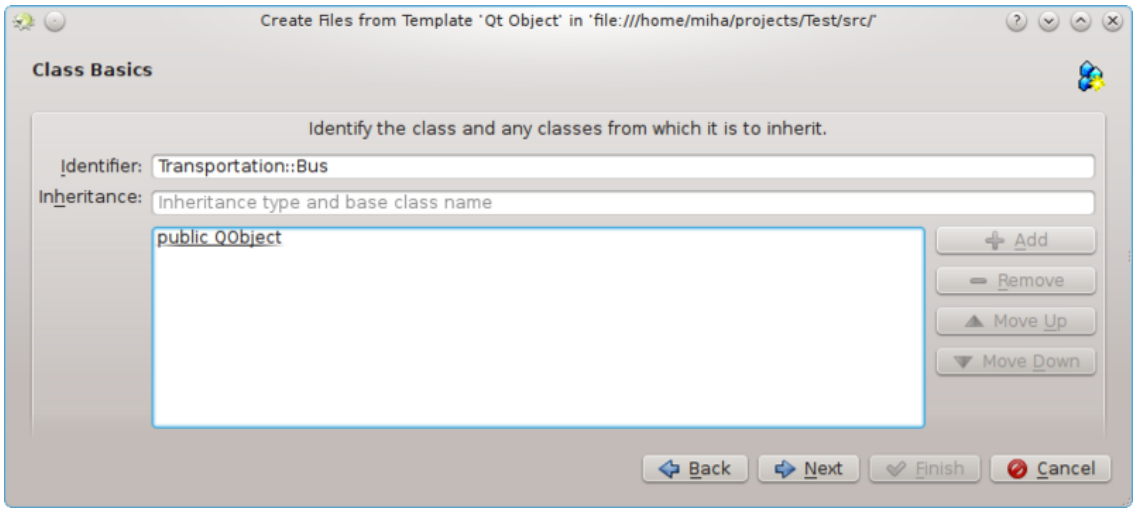
O KDevelop usa modelos para gerar ficheiros de código-fonte e para evitar escrever código repetido.

4.1 Criar uma nova classe

O uso mais comum para a geração de código é provavelmente a criação de classes novas. Para criar uma classe nova num projecto existente, carregue com o botão direito sobre a pasta de um projecto e escolha **Criar a Partir de um Modelo...**. A mesma janela poderá ser acedida se carregar em **Ficheiro** → **Novo a Partir do Modelo...**, mas usar uma pasta de projecto terá a vantagem de definir um URL de base para os ficheiros de saída. Escolha `Classe` na área de selecção da categoria e a linguagem de programação e modelo nas duas outras áreas. Depois de ter seleccionado um modelo de classes, terá de indicar os detalhes da nova classe.

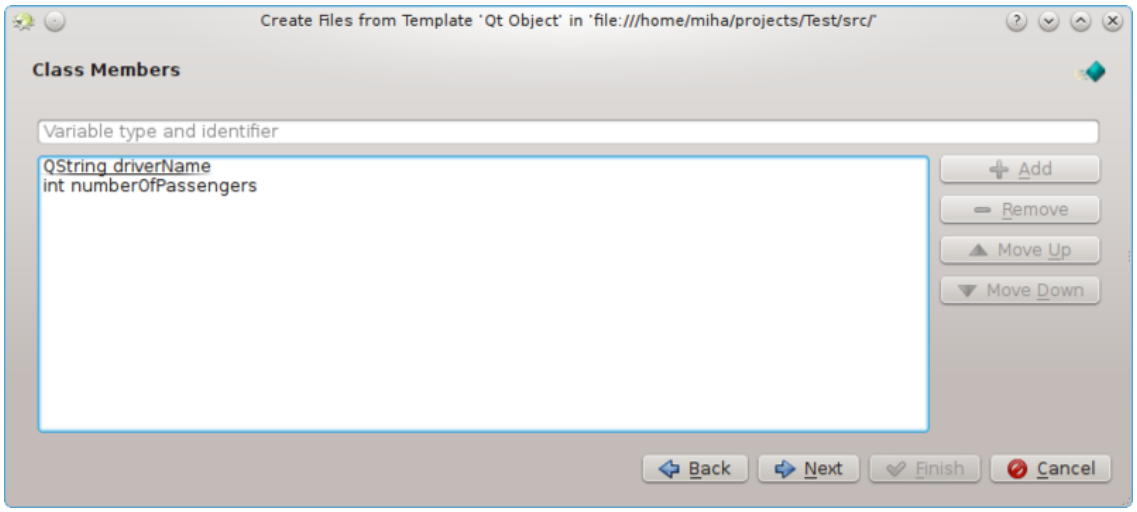


Primeiro tem de indicar um identificador para a nova classe. Poderá ser um nome simples (como `Autocarro`) ou um identificador completo com espaços de nomes (do tipo `Transporte::Autocarro`). No último caso, o KDevelop irá processar o identificador e irá separar os espaços de nomes do nome propriamente dito. Na mesma página, poderá adicionar classes de base para a classe nova. Poderá ver que alguns modelos escolhem uma classe de base própria, estando à vontade para removê-la e/ou adicionar outras. Deverá escrever aqui toda a instrução de herança, a qual depende da linguagem, como por exemplo `public ClasseBase` no C++, `extends ClasseBase` no PHP ou simplesmente o nome da classe no Python.



Na página seguinte, é-lhe oferecida uma selecção dos métodos virtuais de todas as classes herdadas, assim como os construtores, destrutores e operadores predefinidos. Se assinalar a opção associada à assinatura de um dado método, irá implementar esse método na nova classe.

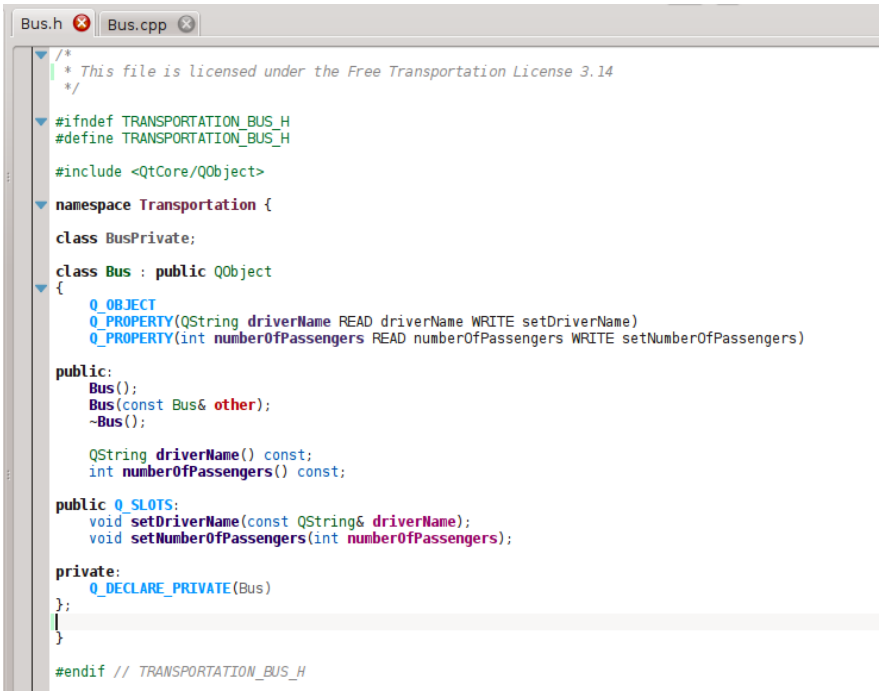
Se carregar em **Seguinte**, irá aparecer uma página onde poderá adicionar membros a uma classe. Dependendo do modelo escolhido, estas poderão aparecer na classe nova como variáveis-membro, ou então o modelo poderá criar propriedades com métodos 'get' e 'set' para elas. Numa linguagem em que os tipos de variáveis tenham de ser declarados, como o C++, terá de indicar tanto o tipo como o nome do membro, como por exemplo `int idade` ou `QString nome`. Noutras linguagens, poderá ignorar o tipo, mas é uma boa prática defini-lo à mesma, dado que o modelo seleccionado poderá à mesma tirar partido dele.



Nas páginas seguintes, poderá escolher uma licença para a sua nova classe, definir quaisquer opções personalizadas que sejam necessárias para o modelo seleccionado e configurar os locais de saída de todos os ficheiros gerados. Ao carregar em **Terminar**, irá terminar o assistente e irá criar a nova classe. Os ficheiros gerados serão abertos no editor, para que possa logo começar a adicionar código.

Depois de criar uma nova classe em C++, ser-lhe-á dada a opção para adicionar a classe a um dado alvo do projecto. Escolha um alvo na página da janela ou feche a mesma e adicione manualmente os ficheiros a um dado alvo.

Se escolheu o modelo `Qt Object`, assinalou alguns métodos predefinidos e tiver adicionado duas variáveis-membro, o resultado deverá ser algo semelhante à imagem seguinte.



Poderá ver que os membros de dados são convertidos em propriedades do Qt, com funções de acesso e as macros Q_PROPERTY. Os argumentos das funções 'set' são até passados por referência constante onde for apropriado. Para além disso, é declarada uma classe privada e um ponteiro privado, criado através do Q_DECLARE_PRIVATE. Tudo isto é feito pelo modelo; se escolher outro modelo no primeiro passo, poderá mudar o resultado por completo.

4.2 Criar um novo teste unitário

Ainda que a maioria das plataformas de testes necessitem que cada teste seja também uma classe, o KDevelop inclui um método para simplificar a criação de testes unitários. Para criar um novo teste, carregue com o botão direito sobre a pasta de um projecto e seleccione **Criar a Partir de um Modelo...**. Na página de selecção do modelo, escolha o `Teste` como categoria, depois escolha a sua linguagem de programação e modelo e carregue em **Seguinte**.

Ser-lhe-á pedido o nome do teste e uma lista de casos de testes. Para os casos de testes, só terá de indicar uma lista de nomes. Algumas plataformas de testes unitários, como o PyUnit e o PHPUnit, necessitam que os casos de testes comecem por um dado prefixo especial. No KDevelop, o modelo é responsável pela adição do prefixo, pelo que não o terá de indicar aqui nos casos de teste. Depois de carregar em **Seguinte**, indique a licença e os locais de saída dos ficheiros gerados, para que o teste seja depois criado.

Os testes unitários criados desta forma não serão adicionados a nenhum alvo automaticamente. Se estiver a usar o CTest ou outra plataforma de testes, certifique-se que adiciona os novos ficheiros a um alvo.

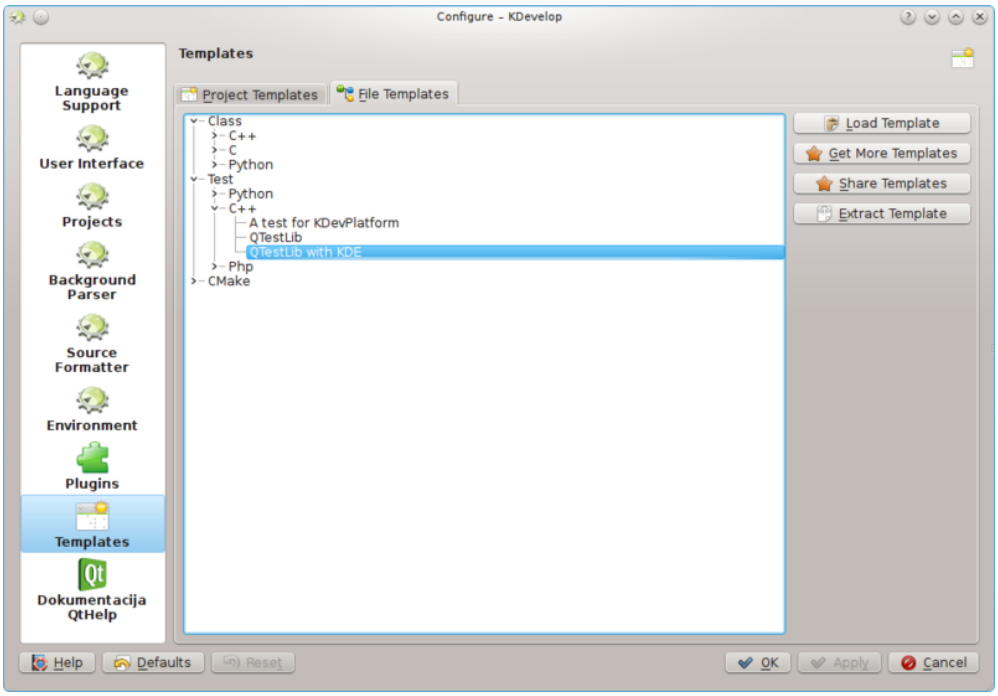
4.3 Outros ficheiros

Embora as classes e testes unitários recebam uma atenção especial ao gerar o código a partir dos modelos, o mesmo método poderá ser usado para qualquer tipo de ficheiro de código-fonte. Por exemplo, uma pessoa poderá usar um modelo para um módulo de pesquisa do CMake ou para um ficheiro '.desktop'. Isto pode ser feito se escolher **Criar a Partir de um Modelo...** e seleccionar a categoria e modelo desejados. Se a categoria seleccionada não for nem a `Classe` nem a `Teste`,

só terá a opção para escolher a licença, opções específicas do modelos e o local dos ficheiros de saída. Como nas classes e testes, ao terminar o assistente, irá gerir os ficheiros e abri-los no editor.

4.4 Gerir os modelos

No assistente **Ficheiro** → **Novo a Partir de um Modelo...**, poderá também transferir modelos de ficheiros adicionais a partir do botão **Obter Mais Modelos...**. Isto irá abrir uma janela para Obter Coisas Novas, onde poderá instalar mais modelos, assim como actualizá-los e removê-los. Existe também um módulo de configuração para os modelos, que poderá ser atingido se carregar em **Configuração** → **Configurar o KDevelop** → **Modelos**. A partir daí, poderá gerir tanto os modelos de ficheiros (explicados acima) como os modelos de projectos (usados para criar novos projectos).



Obviamente, se nenhum dos modelos disponíveis se adequar ao seu projecto, poderá sempre criar novos. A forma mais simples será provavelmente copiar e modificar um modelo existente, embora exista um breve tutorial e um documento de referência mais extenso para o ajudar. Para copiar um modelo instalado, abra o gestor de modelos com a opção **Configuração** → **Configurar o KDevelop...** → **Modelos**, seleccione o modelo que deseja copiar e depois carregue no botão **Extrair o Modelo**. Seleccione uma pasta de destino, carregue depois em **OK** e então o conteúdo do modelo será extraído para a pasta seleccionada. Aí, poderá então editar o modelo, abrindo os ficheiros extraídos e modificando-os. Depois de terminar, poderá importar o seu novo modelo no KDevelop, abrindo para tal o gestor de modelos, activando a página apropriada (ou os **Modelos de Projectos** ou os **Modelos de Ficheiros**) e carregando em **Carregar um Modelo**. Abra o ficheiro de descrição do modelo, que será o que tiver o sufixo `.kdevtemplate` ou `.desktop`. O KDevelop irá comprimir os ficheiros num pacote de modelo e irá importar o mesmo.

> NOTA
> Ao copiar um modelo existente, certifique-se que lhe muda o nome antes de o importar de novo. Caso contrário, irá substituir o modelo antigo ou irá terminar com dois modelos de nome igual. Para mudar o nome de um modelo, mude o nome do ficheiro de descrição para algo único (embora mantendo o sufixo) e mude o item `Name` no ficheiro da descrição.

Se quiser criar um modelo do zero, poderá começar com um modelo de classe de exemplo em C++, para tal criando um novo projecto e seleccionando o projecto do `Modelo de Classe em C++` na categoria `KDevelop`.

Capítulo 5

Compilar os projectos com Makefiles personalizados

Muitos projectos descrevem como é que os ficheiros de código devem ser compilados (e quais os ficheiros que terão de ser recompilados assim que um ficheiro de código ou de inclusão mudar), usando os ficheiros Makefile que são interpretados pelo programa **make** (veja, por exemplo, o 'make' da GNU). Para os projectos simples, é normalmente muito simples configurar um destes ficheiros à mão. Os projectos maiores normalmente integram os seus ficheiros Makefile com as **'autotools' da GNU** (autoconf, autoheader, automake). Nesta secção, iremos assumir que você tem um Makefile para o seu projecto e que quer indicar ao KDevelop como interagir com ele.

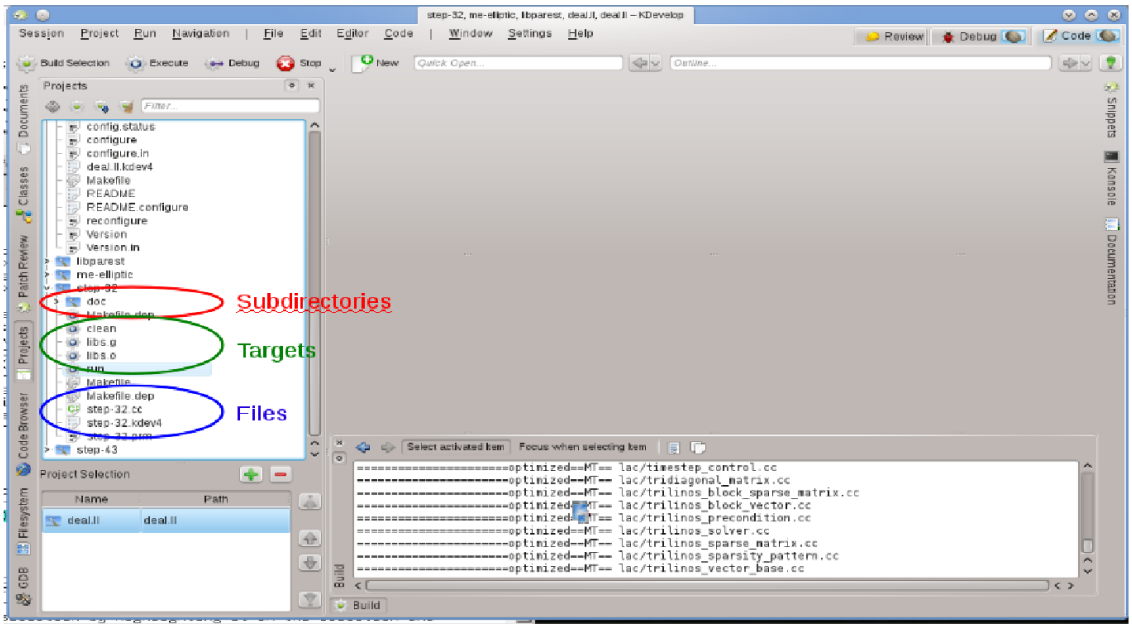
> NOTA
> O KDevelop 4.x não suporta nada das **'autotools' da GNU** na altura em que esta secção foi escrita. Se o seu projecto as usar, terá de correr o `./configure` ou qualquer um dos outros comandos relacionados à mão, numa linha de comandos. Se quiser fazer isto dentro do KDevelop, abra a ferramenta do **Konsole** (se necessário, adicione-a ao perímetro da janela principal, usando a opção **Janelas** → **Adicionar uma área de ferramentas**) que lhe dará uma janela com linha de comandos e poderá então executar o `./configure` a partir da linha de comandos nesta janela.

O primeiro passo é indicar ao KDevelop quais são os alvos nos seus ficheiros Makefile. Existem duas formas de o fazer: se seleccionar os alvos do Makefile individualmente e escolher uma lista dos que deseja compilar com mais frequência. Em ambas as abordagens, abra a ferramenta de **Projectos**, carregando no botão de **Projectos** no perímetro da janela principal do KDevelop (se não tiver este botão, veja como adicionar um botão destes aí). A janela da ferramenta de **Projectos** tem duas partes: a metade superior — chamada **Projectos** — apresenta todos os seus projectos e permite-lhe expandir as árvores de pastas subjacentes. A metade inferior — chamada **Selecção dos Projectos** — apresenta um sub-conjunto desses projectos que serão compilados ao escolher o item do menu **Projecto** → **Compilar a selecção** ou carregar em **F8**; iremos voltar a esta parte mais em baixo.

5.1 Compilar os alvos individuais do Makefile

Na parte superior da área do projecto, expanda a sub-árvore de um projecto, por exemplo o projecto onde deseja executar um alvo em particular do Makefile. Isto dar-lhe-á ícones para (i) as pastas sob este projecto, (ii) os ficheiros na pasta de topo deste projecto, (iii) os alvos do Makefile que o KDevelop consegue identificar. Estas categorias aparecem na imagem à direita. Lembre-se que o KDevelop *compreende* a sintaxe do Makefile até um certo ponto e, como tal, consegue

apresentar-lhe os alvos definidos nesse Makefile (ainda que esta compreensão possa ter os seus limites, caso os alvos sejam compostos ou implícitos.



Para compilar qualquer um dos alvos aqui apresentados, carregue nele com o botão direito do rato e seleccione **Compilar**. Por exemplo, se fizer isto com o alvo 'clean' (limpar), irá simplesmente executar o comando 'make clean'. Poderá ver isto a acontecer na sub-janela **Compilação** que aparece, mostrando-lhe o comando e o seu resultado. (Esta janela corresponde à ferramenta para **Compilar**, pelo que poderá fechar e voltar a abrir a janela com o botão de ferramentas **Compilar** no perímetro da janela principal. Este aparece na parte inferior direita da imagem.)

5.2 Seleccionar uma colecção de alvos do Makefile para uma compilação repetida

Se carregar com o botão direito em alvos individuais do Makefiles, sempre que quiser compilar algo, irá perder tempo precioso. Em vez disso, será bom ter alvos individuais para um ou mais projectos da sessão que possa então compilar de forma repetida sem muito trabalho com o rato. Aí é onde o conceito das 'Selecções de alvos de compilação' pode ajudar: é uma colecção de alvos dos ficheiros Makefile que são executados um a seguir ao outro quando carregar no botão **Compilar a selecção** na lista de botões do topo, seleccionar a opção do menu **Projecto** → **Compilar a selecção** ou carregar na tecla de função **F8**.

A lista com os alvos seleccionados da Makefile aparece na metade inferior da área de **Projectos**.

Por omissão, a selecção contém todos os projectos, mas você poderá alterar isso. Por exemplo, se a sua lista de projectos tiver três destes (uma biblioteca de base L e duas aplicações A e B), mas se só estiver a trabalhar de momento no projecto A, poderá querer remover o projecto B da selecção, seleccionando-o nessa lista e carregando no botão . Para além disso, poderá querer garantir que a biblioteca L é compilada antes do projecto A, movendo os itens da selecção para cima ou para baixo com os botões à direita da lista. Também poderá obter um alvo da Makefile em particular para a selecção se carregar com o botão direito sobre ela e seleccionar **Adicionar ao conjunto de compilação** ou simplesmente seleccioná-lo e carregando em , logo por cima da lista de alvos seleccionados.

O KDevelop permite-lhe configurar o que desejar sempre que compilar a selecção. Para tal, use a opção do menu **Projecto** → **Abrir a configuração**. Aí, poderá por exemplo seleccionar o número de tarefas simultâneas que o 'make' deverá executar — se o seu computador tiver, por exemplo, 8 processadores, então poderá ser útil indicar 8 neste campo. Nesta janela, o **Alvo predefinido do 'make'** é um alvo do Makefile usado para *todos* os alvos da selecção.

5.3 O que fazer com as mensagens de erro

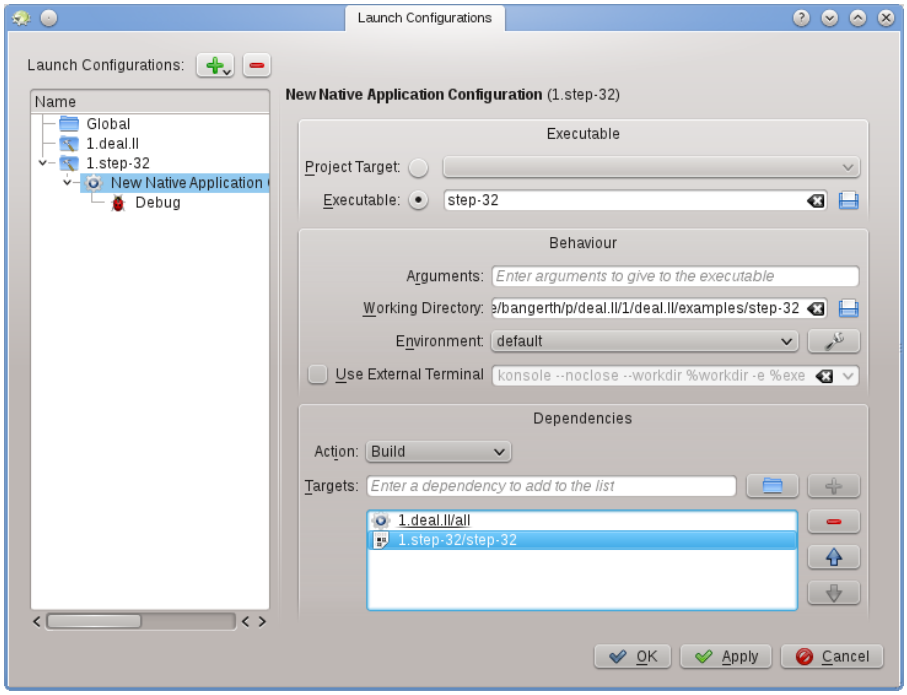
Se o compilador encontrar uma mensagem de erro, basta carregar na linha com a mensagem de erro para que o editor vá para a linha (e, se possível, a coluna) onde foi comunicado o erro. Dependendo da mensagem de erro, o KDevelop poder-lhe-á oferecer várias acções possíveis para corrigir o erro, como por exemplo declarar uma variável previamente ainda por declarar, caso seja encontrado um símbolo desconhecido.

Capítulo 6

Executar os programas no KDevelop

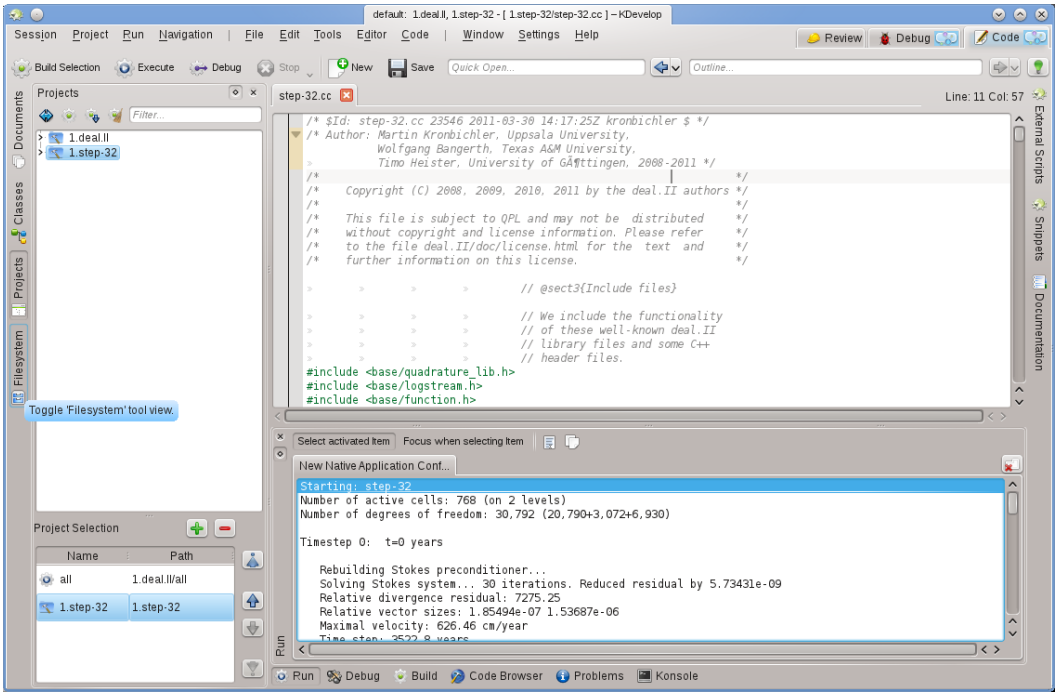
Assim que tiver compilado um programa, irá querer executá-lo. Para tal, é necessário configurar *Lançamentos* para os seus projectos. Um *Lançamento* consiste no nome de um executável, um conjunto de parâmetros da linha de comandos e um ambiente de execução (como por exemplo 'executar este programa num terminal' ou 'executar este programa no depurador').

6.1 Configurar os lançamentos no KDevelop



Para configurar isto, vá à opção do menu **Executar** → **Configurar os lançamentos**, seleccione o projecto para o qual deseja adicionar um lançamento e carregue no botão . Depois, indique o nome do executável e o local onde seja executá-lo. Se a execução do programa depender da compilação do executável e/ou das suas bibliotecas em primeiro lugar, então podê-las-á adicionar à lista no fundo: seleccione **Compilar** no menu e depois carregue no símbolo à direita

do campo de texto e seleccione o alvo que deseja ter compilado. No exemplo acima, foi seleccionado o alvo **all** (tudo) do projecto `1.jogada.II` e `passo-32` do projecto `1.passo-32` para se certificar que tanto a biblioteca de base como o programa foram compilados e estão actualizados antes de executar o programa em si. Já que está aqui, poderá também configurar um lançamento de depuração, carregando para tal no símbolo **Depuração** e adicionando o nome do programa de depuração; se este for o depurador predefinido do sistema (isto é o gdb no Linux®), então não terá de efectuar este passo.



Poderá agora tentar executar o programa: Seleccione **Executar** → **Executar o Lançamento** a partir do menu da janela principal do KDevelop (ou carregar em **Shift-F9**), para que o seu programa se execute numa sub-janela separada do KDevelop. A imagem acima mostra o resultado: a nova sub-janela da ferramenta **Executar**, no fundo, mostra o resultado do programa que está a ser executado, neste caso, do programa `passo-32`.

> NOTA
> Se tiver configurado vários lançamentos, poderá escolher qual é que deseja executar quando carregar em **Shift-F9**, indo à opção **Executar** → **Configuração de Lançamento Actual**. Existe uma forma não-óbvia de editar o nome de uma configuração: na janela que obtém quando seleccionar a opção **Executar** → **Configuração de Lançamento Actual**, faça duplo-click sobre o nome da configuração na árvore da esquerda, a qual lhe permitirá editar o nome da configuração.

6.2 Algumas combinações de teclas úteis

Executar um programa	
F8	Compilar (invocar o 'make')
Shift-F9	Executar

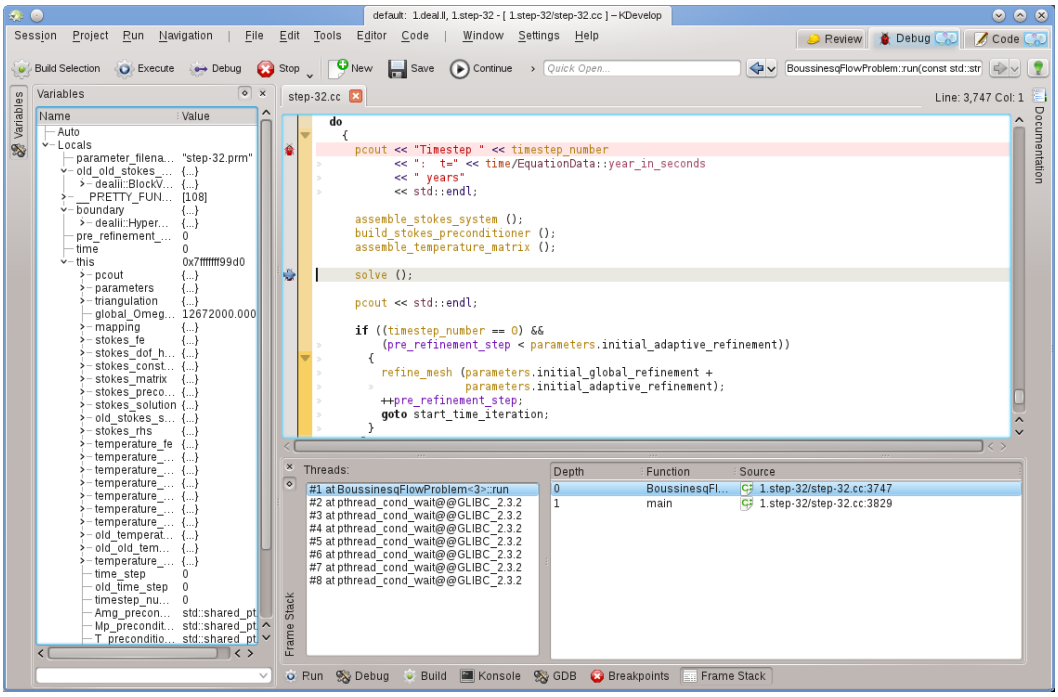
F9	Executar o programa no depurador; poderá querer definir pontos de paragem de antemão; por exemplo, se carregar com o botão direito do rato numa linha em particular do código-fonte

Capítulo 7

Depurar os programas no KDevelop

7.1 Executar um programa no depurador

Assim que tiver um lançamento configurado (veja como Executar os programas), também o poderá executar num depurador: Seleccione o item do menu **Executar** → **Depurar o Lançamento** ou carregue em **F9**. Se estiver familiarizado com o gdb, o efeito é o mesmo que iniciar o gdb com o nome do executável indicado na configuração do lançamento e depois dizer para o executar. Isto significa que, caso o programa invoque o `abort()` algures (isto é quando você chegar a uma asserção mal-sucedida) ou se tiver um erro de segmentação (protecção de memória), então o depurador irá parar. Por outro lado, se o programa chegar ao fim (tendo ou não feito a coisa certa), então o depurador não irá parar por si só antes que o programa termine. No último caso, irá querer definir um ponto de paragem sobre todas essas linhas da sua base de código onde deseja que o depurador pare, antes de executar o lançamento de depuração. Podê-lo-á fazer com a opção do menu **Executar** → **Comutar o ponto de paragem** ou se carregar com o botão direito sobre uma linha e seleccionar a opção **Comutar o Ponto de Paragem** do menu de contexto.

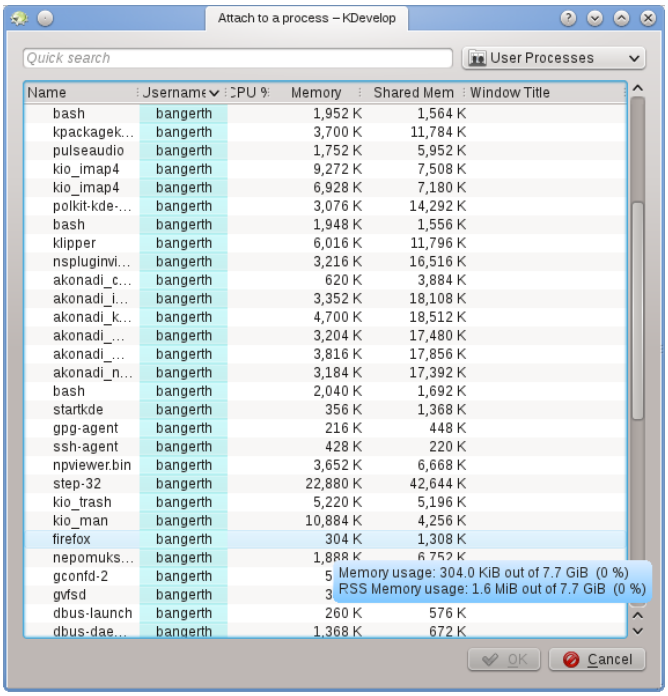


A execução de um programa no depurador irá colocar o KDevelop num modo diferente: irá substituir todos os botões de 'Ferramentas' no perímetro da janela principal por outros que sejam apropriados para a edição. Poderá ver qual dos modos em que se encontra se olhar para o canto superior direito da janela: existem páginas chamadas **Rever**, **Depuração** e **Código**; se carregar nelas, poderá mudar para qualquer um dos três modos; cada modo tem um conjunto de áreas de ferramentas próprio, o qual poderá configurar da mesma forma que foi feito para as ferramentas de **Código** na secção Ferramentas e janelas.

Assim que o depurador parar (num ponto de paragem ou num ponto em que a função `abort()` seja chamada), poderá inspeccionar uma grande quantidade de informação acerca do seu programa. Por exemplo, na imagem acima, foi seleccionada a **Pilha de Chamadas** no fundo (algo equivalente aos comandos do gdb 'backtrace' e 'info threads') que mostra as várias tarefas em execução do seu programa à esquerda (aqui num total de 8) e como é que a execução chegou ao ponto de paragem actual à direita (aqui: o `main()` invocou o `executar()`; a lista seria maior se tivesse parado numa função chamada pelo próprio `executar()`). À esquerda, poderá inspeccionar as variáveis locais, incluindo o objecto actual (o objecto referenciado pela variável `this`).

A partir daqui, existem várias possibilidades disponíveis para si: poderá executar a linha actual (**F10**, equivalente ao comando do gdb 'next'), ir para dentro das funções (**F11**, correspondendo ao comando do gdb 'step') ou executar até ao fim da função (**F12**, equivalente ao comando do gdb 'finish'). Em cada passo, o KDevelop actualiza as variáveis apresentadas à esquerda para os seus valores actuais. Poderá também passar o rato sobre um símbolo no seu código, isto é uma variável; o KDevelop irá então mostrar o valor actual desse símbolo e oferecer-se-á para parar o programa na próxima vez que o valor desta variável mudar. Se conhecer o gdb, também poderá carregar no botão da ferramenta **GDB** no fundo e ter a possibilidade de introduzir directamente comandos do gdb, por exemplo para alterar o valor de uma variável (possibilidade para a qual não existe de momento outra forma alternativa).

7.2 Associar o depurador a um processo em execução



Algumas vezes, uma pessoa poderá querer depurar um programa que já está em execução. Um cenário para isso será a depuração de vários programas em paralelo com o MPI ou para depurar

um programa que se encontra há muito em segundo plano. Para tal, vá à opção do menu **Executar** → **Anexar ao Processo**, a qual irá abrir uma janela como a anterior. Irá querer seleccionar o programa que corresponde ao seu projecto aberto de momento no KDevelop - neste caso, seria o programa 'passo-32'.

Esta lista de programas poderá ser confusa porque é normalmente muito longa, como acontece no caso daqui. Poderá simplificar a sua vida se for à lista no canto superior direito da janela. O valor por omissão é **Processos do utilizador**, isto é todos os programas que são executados por qualquer um dos utilizadores autenticados de momento nesta máquina (se este for o seu computador pessoal ou portátil, provavelmente você será o único utilizador de facto, para além do 'root' e das várias contas de serviços); a lista não inclui os processos executados pelo utilizador 'root', contudo. Poderá limitar a lista se escolher a opção **Processos próprios**, removendo todos os programas executados pelos outros utilizadores. Melhor ainda, seleccione a opção **Apenas os programas**, a qual retira muitos dos processos que estão a ser executados com o seu nome, mas com os quais não interage normalmente, como o gestor de janelas, as tarefas de segundo plano e assim por diante, as quais não são normalmente candidatas para a depuração.

Assim que tiver seleccionado um processo, ao associar-se a ele irá entrar no modo de depuração do KDevelop, abrir todas as áreas de ferramentas de depuração e parar o programa na posição em que se encontrava quando se associou a ele. Aí poderá querer definir pontos de paragem, pontos de visualização ou tudo o que necessitar e ainda continuar a execução do programa, indo para a opção do menu **Executar** → **Continuar**.

7.3 Algumas combinações de teclas úteis

Depuração	
F10	Avançar sobre ('next' do 'gdb')
F11	Avançar para ('step' do 'gdb')
F12	Avançar para fora ('finish' do 'gdb')

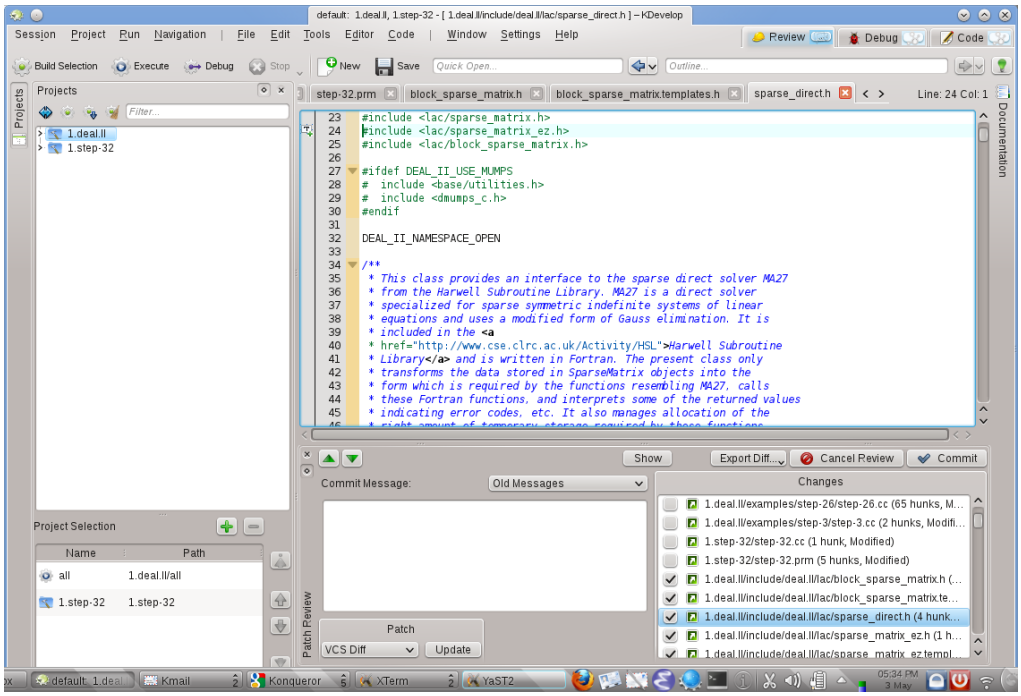
Capítulo 8

Lidar com sistemas de controlo de versões

Se estiver a lidar com projectos maiores, será provável que o código-fonte seja gerido por um sistema de controlo de versões como o subversion ou o git. A seguinte descrição será feita com o **subversion** em vista, mas será igualmente válida se quiser usar o **git** ou outro sistema de controlo de versões suportado qualquer.

Repare primeiro que, se a pasta na qual se encontra um projecto estiver sob controlo de versões, o KDevelop irá descobrir automaticamente. Por outras palavras: Não é necessário que indique ao KDevelop para extrair ele próprio uma cópia ao configurar o seu projecto; é suficiente apontar o KDevelop para uma pasta onde já tenha extraído previamente uma cópia do repositório. Se tiver uma dessas pastas sob controlo de versões, abra a área de ferramentas dos **Projectos**. Aí, existe um conjunto de coisas que poderá fazer:

- Se a sua pasta se tornou desactualizada, podê-la-á actualizar a partir do repositório: Carregue no nome do projecto com o botão direito do rato, vá ao menu **Subversion** e seleccione **Actualizar**. Isto irá obter actualizações de todos os ficheiros que pertençam a este projecto e que digam respeito ao repositório.
- Se quiser restringir esta acção apenas às sub-pastas ou ficheiros individuais, então expanda a árvore deste projecto para o nível que desejar e carregue com o botão direito sobre uma sub-pasta ou ficheiro, fazendo o mesmo que se descreveu acima.



- Se tiver editado um ou mais ficheiros, expanda a área do projecto até à pasta onde se encontram estes ficheiros e carregue com o botão direito sobre a pasta. Isto oferecer-lhe-á um item do menu **Subversion** que lhe oferece diferentes opções. Escolha a opção **Comparar com a base** para ver as diferenças entre a versão que tem editada e a versão no repositório que extraiu anteriormente (a versão de 'base'). A janela resultante irá mostrar as 'diferenças' de todos os ficheiros nesta pasta.

- Se só editou um único ficheiro, poderá também obter o menu **Subversion** para este ficheiro, bastando para tal carregar com o botão direito sobre o ficheiro correspondente na área do projecto. Ainda mais simples, basta carregar com o botão direito sobre a área do **Editor**, na qual tenha aberto este ficheiro, obtendo também esta opção do menu.

- Se quiser enviar para o servidor um ou mais ficheiros editados, carregue com o botão direito sobre um ficheiro individual, sub-pasta ou sobre o projecto todo e seleccione a opção **Subversion** → **Enviar**. Isto fará mudar o modo para **Revisão**, o terceiro modo que existe para além do **Código** e **Depuração** no canto superior direito da janela principal do KDevelop. A imagem à direita mostra-lhe como isto fica. No modo de **Revisão**, a parte superior mostra-lhe as diferenças para a sub-pasta/projecto inteiro e cada um dos ficheiros individuais alterados com as alterações realçadas (veja as várias páginas nesta parte da janela). Por omissão, todos os ficheiros alterados estão no conjunto de alterações que estará prestes a enviar, mas poderá desligar alguns dos ficheiros, caso as suas modificações não estejam relacionadas com o que deseja enviar. No exemplo à direita, foi desligado o ficheiro `passo-32.cc` e `passo-32.prm` porque as alterações destes ficheiros não têm nada a ver com as outras que foram feitas no projecto e não se pretende para já enviá-las (poder-se-á pensar nisso num envio em separado). Depois de rever as alterações, poderá introduzir uma mensagem de envio no campo de texto e carregar em **Enviar** à direita para enviar tudo o que desejar.

- Assim como na visualização das diferenças, se quiser enviar um único ficheiro, também poderá carregar com o botão direito na janela do editor para obter a opção do menu **Subversion** → **Enviar**.

Capítulo 9

Personalizar o KDevelop

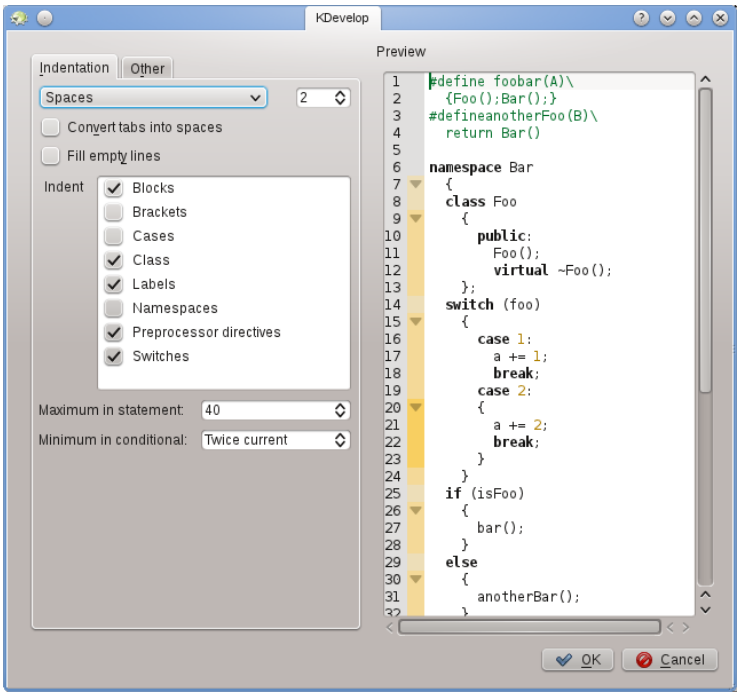
Existem alturas em que poderá querer alterar a aparência ou comportamento predefinidos do KDevelop; por exemplo, se tiver habituado a combinações de teclas diferentes ou porque o seu projecto necessita de um estilo de indentação diferente para o código-fonte. Nas seguintes secções, iremos discutir de forma breve as diferentes formas com que pode personalizar o kdevelop; para esses fins.

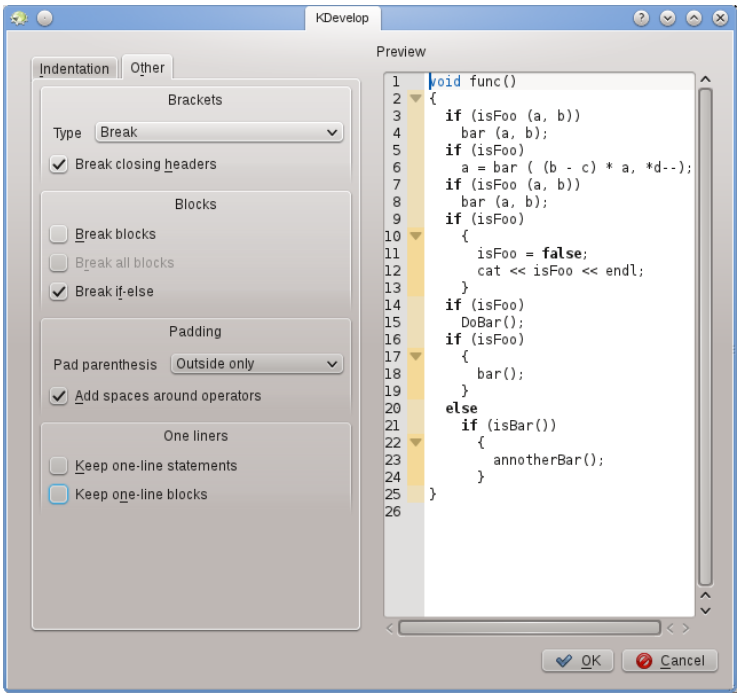
9.1 Personalizar o editor

Existe um conjunto de coisas úteis que poderá configurar no editor incorporado do KDevelop. A mais universal será activar a numeração de linhas com a opção do menu **Editor** → **Ver** → **Mostrar os números de linha**, facilitando a correspondência das mensagens de erro do compilador ou do depurador com os locais do código. No mesmo submenu, poderá também querer activar o *Contorno de ícones* - uma coluna à esquerda do seu código na qual o KDevelop irá mostrar ícones como os de existência de pontos de paragem na linha actual.

9.2 Personalizar a indentação do código

Muitos de nós gostamos do código formatado de uma determinada forma. Muitos projectos também obrigam a um dado estilo de indentação em particular. Alguns deles poderão não corresponder aos estilos predefinidos do KDevelop. Contudo, isto pode ser personalizado: vá à opção do menu **Configuração** → **Configurar o KDevelop**, depois carregue no **Formatador de Código** à esquerda. Poderá então escolher um dos estilos predefinidos de indentação que são vulgarmente usados ou ainda definir o seu próprio, adicionando um novo estilo e depois editando-o. Poderá não haver uma forma de recriar exactamente o estilo com que o código do seu projecto foi indentado no passo, mas poderá aproximar-se o suficiente se usar a configuração de um novo estilo; é demonstrado um exemplo nas duas imagens em baixo.


KDevelop
Indentation
Other
Spaces
2
Convert tabs into spaces
Fill empty lines
Indent
Blocks
Brackets
Cases
Class
Labels
Namespaces
Preprocessor directives
Switches
Maximum in statement:
40
Minimum in conditional:
Twice current
Preview
OK
Cancel


KDevelop
Indentation
Other
Brackets
Type
Break
Break closing headers
Blocks
Break blocks
Break all blocks
Break if-else
Padding
Pad parenthesis
Outside only
Add spaces around operators
One liners
Keep one-line statements
Keep one-line blocks
Preview
OK
Cancel

NOTA
Com o **KDevelop 4.2.2**, poderá criar um novo estilo para um tipo MIME em particular (isto é para os ficheiros de inclusão em C++), mas este estilo poderá não aparecer na lista de estilos possíveis para outros tipos MIME ((isto é para os ficheiros de código em C++) ainda que pudesse ser útil usar o mesmo estilo para ambos os tipos de ficheiros. Nesse caso, terá de definir o estilo duas vezes, uma para os ficheiros de inclusão e outra para os de código. Isto foi comunicado como o erro 272335 do KDevelop.

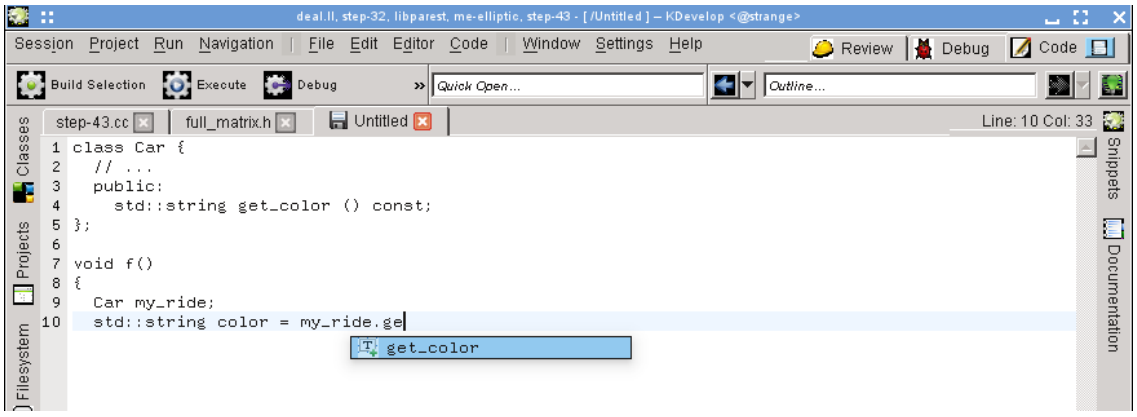
9.3 Personalizar os atalhos de teclado

O KDevelop tem uma lista quase ilimitada de combinações de teclas (algumas delas encontram-se nas 'secções de combinações úteis de teclas' de vários capítulos neste manual) que poderão ser alteradas a seu gosto no menu **Configuração** → **Configurar os Atalhos**. No topo da janela, poderá indicar uma palavra a pesquisar em que só irão aparecer os comandos que corresponderem; aí, poderá editar a combinação de teclas que estará associada a esse comando.

Duas que são consideradas muito úteis para alterar são associar o **Alinhar** à tecla **Tab** (muitas pessoas não introduzem tabulações à mão e assim preferem que o editor escolha a disposição do mesmo; com o atalho alterado, ao carregar em **Tab**, fará com que o KDevelop indente/retire a indentação/alinhe o código). A segunda é associar o **Comutar o Ponto de Paragem** ao **Ctrl-B**, dado ser uma operação bastante frequente.

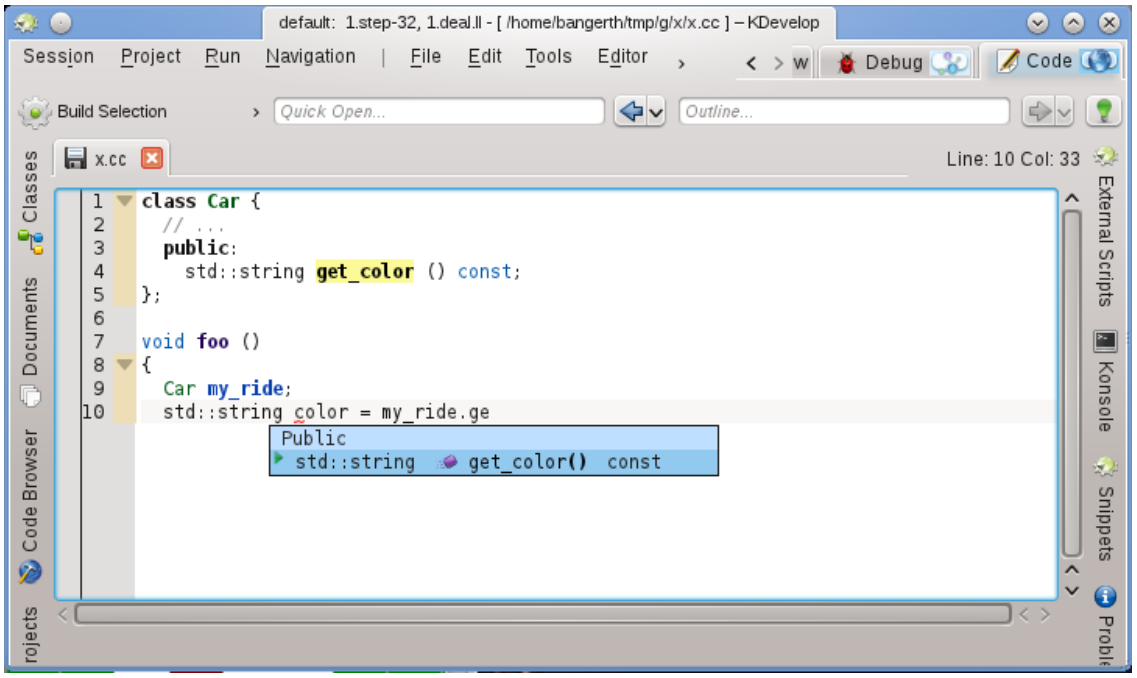
9.4 Personalizar a completação automática do código

A completação do código é discutida na secção deste manual sobre a escrita de código-fonte. No KDevelop, vem de duas origens: o editor e o motor de processamento. O editor (Kate) é um componente do grande ambiente do KDe e fornece a completação automática com base nas palavras que já tiverem sido vista noutras partes do mesmo documento. Essa completação automática poderá ser identificada na dica pelo ícone que a antecede:



A completação de código do editor poderá ser personalizada com a opção **Configuração** → **Configurar o Editor** → **Edição** → **Completação Automática**. Em particular, pode seleccionar quantos caracteres necessita escrever para que a janela de completação automática apareça.

Por outro lado, a completação automática própria do KDevelop é muito mais poderosa, dado que tem em conta a informação semântica acerca do contexto. Por exemplo, sabe que funções-membro deverá oferecer quando escrever `objecto.`, etc., como demonstrado acima:



Esta informação de contexto vem de vários 'plugins' de suporte às linguagens, os quais poderão ser utilizados depois de um dado ficheiro ter sido gravado (para que possa então verificar o tipo de ficheiro e usar o suporte da linguagem correcto).

A completação do KDevelop está configurada para aparecer assim que escrever, praticamente em todo o lado onde seja possível completar algo. Isto é configurável na opção **Configuração** → **Configurar o KDevelop** → **Suporte à Linguagem**. Se não estiver já definido (como deveria, por omissão), certifique-se que a opção **Activar a Invocação Automática** está activa.

O KDevelop tem duas formas de mostrar uma completação: a **Completação Automática Mínima** mostra apenas a informação básica nas dicas de completação (isto é o espaço de nomes, a classe, função ou variável). Isto será semelhante à completação do Kate (exceptuando os ícones).

Por outro lado, a **Completação total** irá também mostrar o tipo de cada item e, no caso das funções, também os argumentos que recebem. Do mesmo modo, se estiver a preencher de momento os argumentos de uma função, a completação total irá ter uma área informativa adicional sobre o cursor que lhe mostrará o argumento actual com que está a lidar.

A completação de código do KDevelop deverá também invocar para o topo e realçar a verde os itens de completação que corresponderem ao tipo esperado, tanto na completação mínima como na total, conhecido como 'melhores ocorrências'.

As três opções possíveis para o nível de completação na janela de configuração são:

- **Sempre a completação mínima**: Nunca mostrar a 'Completação Total'
- **Completação automática mínima**: Só mostrar a 'Completação Total' quando esta tiver sido invocada manualmente (isto é, quando carregar em **Ctrl-Espaço**)
- **Sempre a completação total**: Mostrar sempre a 'Completação Total'

Capítulo 10

Compilar o KDevelop a Partir do Código

Se quiser ter as últimas funcionalidades e correcções de erros, poderá compilar você mesmo o KDevelop a partir do código.

Existe um artigo mais detalhado aqui.

Tenha em atenção que *poderá* compilar uma versão instável. Para ajudar os programadores a corrigir erros, por favor mantenha o `RelWithDebInfo` e comunique os erros em http://bugs.kde.org, seja à mão ou com o Dr. Konqi.

10.1 Requisitos

- kdelibs-devel >= 4.3 - http://www.kde.org
- qt-devel >= 4.5.2 - http://qt-project.org/
- boost-devel >= 1.35 - http://www.boost.org
- g++ >= 4.0 - http://gcc.gnu.org
- CMake >= 2.6.2 - http://www.cmake.org
- qjson-devel

10.2 Instalar para todos os utilizadores

```
mkdir kdevgit
cd kdevgit
git clone git://anongit.kde.org/kdevplatform
git clone git://anongit.kde.org/kdevelop
cd kdevplatform
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ..
make && sudo make install
kbuildsycoca4
cd ../..
```

```
cd kdevelop
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ..
make && sudo make install
kbuildsycoca4
```

10.3 Instalar para o utilizador local

```
mkdir kdevgit
cd kdevgit
git clone git://anongit.kde.org/kdevplatform
git clone git://anongit.kde.org/kdevelop
cd kdevplatform
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DCMAKE_INSTALL_PREFIX=$HOME/ ↩
    kdevelop4 ..
make && make install
```

A seguinte linha é necessária para que o kbuildsycoca4 encontre todos os ficheiros '.desktop'

```
export KDEDIRS=$HOME/kdevelop4:/usr
kbuildsycoca4
cd ../..
cd kdevelop
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DCMAKE_INSTALL_PREFIX=$HOME/ ↩
    kdevelop4 ..
make && make install
kbuildsycoca4
```

Lembre-se por favor: Sempre que fizer alguma actualização de pacotes ou da distribuição que invoque o kbuildsycoca4, terá de executar as seguintes linhas após a actualização:

```
export KDEDIRS=$HOME/kdevelop4:/usr
kbuildsycoca4
```

Capítulo 11

Créditos e Licença

Tradução de José Nuno Pires zepires@gmail.com