Manual do KDevelop

Esta documentação foi convertida a partir da Base de Usuários do KDE, da página KDevelop4/Manual.

Tradução: Marcus Gama
Tradução: André Marcelo Alvarenga

Conteúdo

Resumo

O KDevelop é um Ambiente de Desenvolvimento Integrado para ser usado em uma 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, é executado 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 IDEs modernos. Para grandes projetos 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 sobre o seu código. Por exemplo, imaginemos que um dos arquivos de inclusão do seu projeto declare uma classe

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

e depois no seu programa você tem

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

ele terá recordado que o `meu_carro` da última linha é uma variável do tipo `Carro` e se oferecerá para completar o `co` como `cor()`, uma vez que esta é a única função-membro da classe `Carro` que começa desta forma. Em vez de continuar a escrever, poderá pressionar **Enter** para obter a palavra completa; isto economiza digitação, erros e faz com que você não precise recordar os nomes exatos das centenas ou milhares de funções e classes que compõem os grandes projetos.

Como um segundo exemplo, considere que você possui um código como o seguinte:

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

Se você passar o mouse sobre o símbolo `var` na função `xpto2`, irá obter uma opção para ver todos os usos deste símbolo. Se clicar nele, somente será mostrado 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 clicar com o botão direito no nome da variável, poderá mudar seu nome; 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 mouse sobre um uso desta função; poderá lidar com diferentes tipos de ambientes de compilação e compiladores (por exemplo com o **make** e o **cmake**), entre muitas outras coisas boas que serão discutidas neste manual.

Capítulo 2

Sessões e projetos: O básico do KDevelop

Nesta seção, 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 *projetos*, assim como explicar como poderá configurar os projetos com que deseja trabalhar no KDevelop.

2.1 Terminologia

O KDevelop tem o conceito de *sessões* e *projetos*. Uma sessão contém todos os projetos 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. Você 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 trabalhando num controlador de dispositivo do Linux® que ainda não foi reunido com a árvore de código do 'kernel'.

Assim, pegando o último exemplo, você teria uma sessão no KDevelop com dois projetos: o 'kernel' do Linux® e o controlador do dispositivo. Você desejará agrupá-los numa única sessão (em vez de ter duas sessões com um projeto 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 projeto. Você 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 acessar 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 trabalhando 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 compilando o controlador do dispositivo, caso alguns dos arquivos do 'kernel' tenham mudado).

Finalmente, outro uso para as sessões é se você trabalha tanto na versão atualmente em desenvolvimento de um projeto 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, assim você terá duas sessões com o mesmo conjunto de projetos, mas com pastas diferentes (correspondendo a diferentes ramificações de desenvolvimento).

2.2 Configurar uma sessão e importar um projeto existente

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

Iremos querer preencher esta sessão com projetos que, para agora, consideramos que já existem em algum local (o caso de iniciar os projetos do zero é discutido em outro ponto do manual). Para isso, existem essencialmente dois métodos, dependendo se o projeto já existe em algum local do seu disco ou se precisa de ser transferido a partir de um servidor.

2.2.1 Opção 1: Importar um projeto de um sistema de controle de versões

Iremos considerar que o projeto que desejamos configurar — o 'kernel' do Linux® — reside em algum sistema de controle de versões num servidor, mas que ainda não foi baixado para o seu disco rígido local. Nesse caso, vá para o menu **Projeto** para criar o 'kernel' do Linux® como um projeto dentro da sessão atual e depois siga estes passos:

- Vá para **Projeto** → **Obter projeto** para importar um projeto
- Você terá várias opções para iniciar um projeto novo na sessão atual, dependendo de onde vêm os arquivos 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.
- Considerando que você não possui já uma versão extraída do servidor:
  - Na janela, em **Selecionar origem**, opte por usar o **Do sistema de arquivos**, **Subversion**, **Git**, **GitHub** ou **KDE**
  - Selecione uma pasta de trabalho como destino para onde será extraído o código
  - Escolha a URL da localização no repositório onde se podem obter os arquivos de código
  - Clique em **Obter**. Isto poderá levar bastante tempo, dependendo da velocidade da sua conexão e do tamanho do projeto. Infelizmente, no KDevelop 4.2.x, a barra de progresso não mostra nada de fato, mas você poderá seguir a evolução se olhar periodicamente para o resultado do comando do console

    ```
    du -sk /local/do/projeto/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 *Você precisa indicar uma localização válida para o projeto*; ela poderá ser ignorada sem problemas.

- Será solicitado para que você selecione um arquivo de projeto do KDevelop nesta pasta. Uma vez que provavelmente você não terá ainda nenhum definido, basta clicar em **Seguinte**

- Clique em **Seguinte** de novo
- O KDevelop então solicitará para que você escolha um gerenciador de projeto. Se este projeto usar os arquivos do 'make' do UNIX®, escolha o gerenciador de projetos com 'makefiles' personalizados
- O KDevelop começará então a processar todo o projeto. Mais uma vez, irá levar bastante tempo percorrendo todos os arquivos e a indexando as classes, etc. Na parte inferior direita da janela principal, existe uma barra de progresso que mostra o quanto este processo já percorreu (se tiver vários processadores, você poderá acelerar este processo se for à opção **Configurações** → **Configurar o KDevelop** e selecionar 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 projeto que já exista no seu disco rígido

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

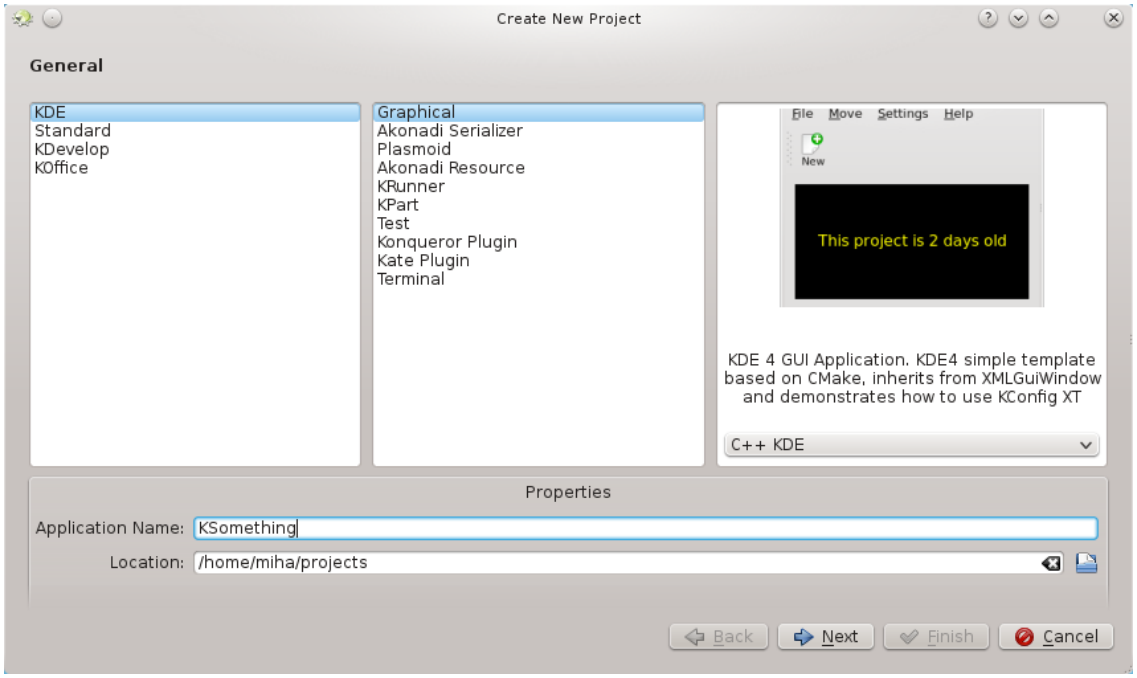
2.3 Configurar uma aplicação como um segundo projeto

A próxima ação que você desejará fazer é configurar outros projetos na mesma sessão. No exemplo acima, poderá querer adicionar o controlador do dispositivo como segundo projeto, o que poderá ser feito usando exatamente os mesmos passos.

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

2.4 Criar projetos do zero

Existe obviamente também a possibilidade de iniciar um novo projeto do zero. Isso pode ser feito usando a opção do menu **Projetos** → **Novo a partir de modelo...**, que exibirá para você uma janela para seleção de modelo. Alguns modelos de projeto são fornecidos com o KDevelop, mas muitos outros estão disponíveis ao instalar o aplicativo KAppTemplate. Selecione o tipo de projeto e linguagem de programação a partir da janela, insira um nome e localização para o seu projeto e clique em **Próximo**.



A segunda página da janela permite que você configure um sistema de controle de versões. Selecione o sistema que deseja usar e preencha as configurações específicas do sistema se necessário. Se você não deseja usar um sistema de controle de versões ou deseja configurá-lo manualmente mais tarde, selecione **Nenhum**. Quando estiver satisfeito com suas opções, pressione **Terminar**.

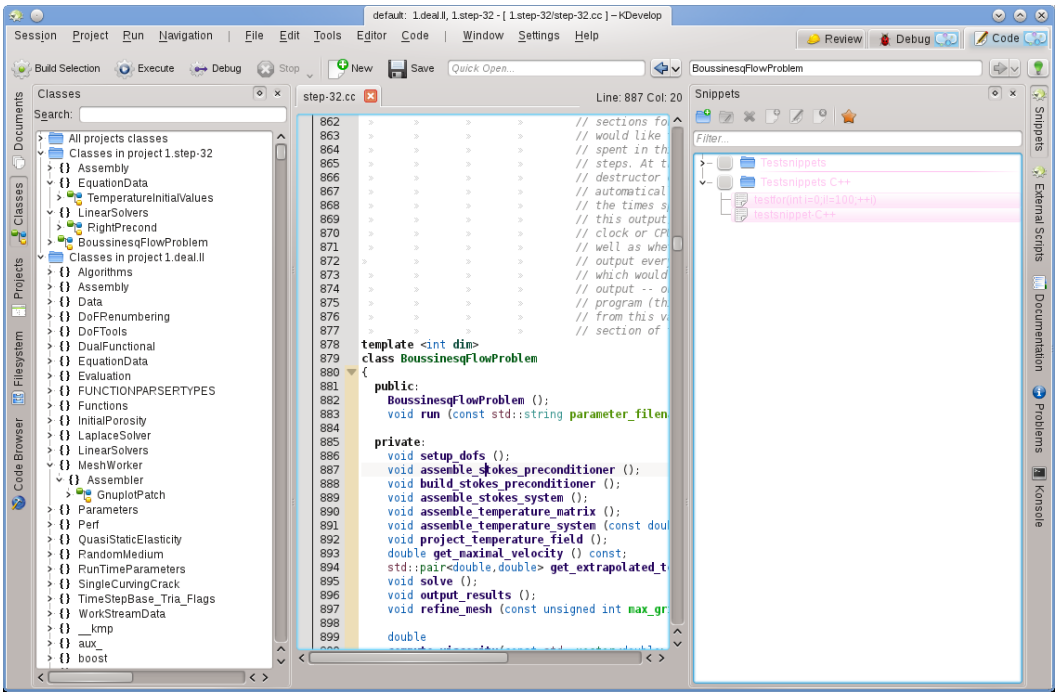
Agora, seu projeto foi criado, de modo que você pode tentar compilá-lo ou instalá-lo. Alguns modelos incluirão comentários no código, ou até mesmo um arquivo README separado, e é recomendável que você leia estas informações inicialmente. Então, você pode iniciar a trabalhar em seu projeto, adicionando as funcionalidades que desejar.

Capítulo 3

Trabalhar com o código-fonte

Além da depuração, a leitura e escrita de código será onde irá gastar mais tempo desenvolvendo as aplicações. Para isso, o KDevelop oferece várias formas de explorar o código-fonte e de tornar mais produtiva a escrita do mesmo. Como será discutido com mais detalhes nas seções a seguir, o KDevelop não é apenas um editor de código — em vez disso, é um sistema de gerenciamento de código que lhe dá diferentes visões sobre a informação extraída dos arquivos que compõem o código-fonte da sua sessão.

3.1 Ferramentas e visões



Para lidar com os projetos, o KDevelop tem o conceito de *ferramentas*. Uma ferramenta oferece uma visão em particular sobre o código ou uma ação a efetuar sobre ele. As ferramentas são representadas como botões em torno da sua janela (com texto vertical ao longo das margens esquerda e direita ou ainda horizontalmente, ao longo da margem inferior). Se clicar nelas, as

mesmas serão expandidas para uma subjanela — uma *área ou visão* — dentro da janela principal; se clicar no botão de ferramentas de novo, a subjanela desaparece.

Para fazer uma subjanela desaparecer, você também poderá clicar no **x** existente no canto superior direito da subjanela

A imagem acima mostra uma seleção em particular das ferramentas, alinhadas ao longo das margens esquerda e direita; na imagem, a ferramenta de **Classes** está aberta à esquerda e os **Trechos** à direita, em conjunto com um editor de um arquivo de código no meio. Na prática, na maior parte do tempo você terá 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 pela primeira vez, você já terá o botão de ferramentas de **Projetos**. Clique nele: irá abrir uma subjanela que mostra os projetos que tiver adicionado à sessão no fundo, assim como uma visão do sistema de arquivos das pastas dos seus projetos no topo.

Existem muitas outras ferramentas que poderá usar com o KDevelop, onde nem todas estarão inicialmente presentes como botões no perímetro. Para adicionar algumas delas, vá para a 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 projetos ou na sua sessão, com todas as suas funções e variáveis-membros. Se clicar em qualquer dos membros, irá abrir um editor de código no local do item onde clicou.
- **Documentos**: Apresenta alguns dos arquivos visitados recentemente, classificados pelo tipo (por exemplo arquivos de código, arquivos de modificações, documentos de texto simples).
- **Navegador de código**: Dependendo da posição do seu cursor num arquivo, esta ferramenta mostra as coisas que estejam relacionadas entre si. Por exemplo, se estiver numa linha `#include`, irá mostrar informações sobre o arquivo que está incluindo, como as classes que estão declaradas nesse arquivo; se estiver numa linha vazia ao nível do arquivo, irá mostrar as classes e funções declaradas e definidas no arquivo atual (tudo como hiperligações: se clicar nelas, irá para o ponto do arquivo onde se encontra de fato 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 arquivos**: Mostra-lhe uma visão em árvore do sistema de arquivos.
- **Documentação**: Permite-lhe procurar nas páginas de manual e em outros documentos de ajuda.
- **Trechos**: Isto fornece sequências de texto que uma pessoa poderá usar quantas vezes quiser e que não terá que escrever sempre. Por exemplo, no projeto 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 uma boa candidato para um trecho.

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

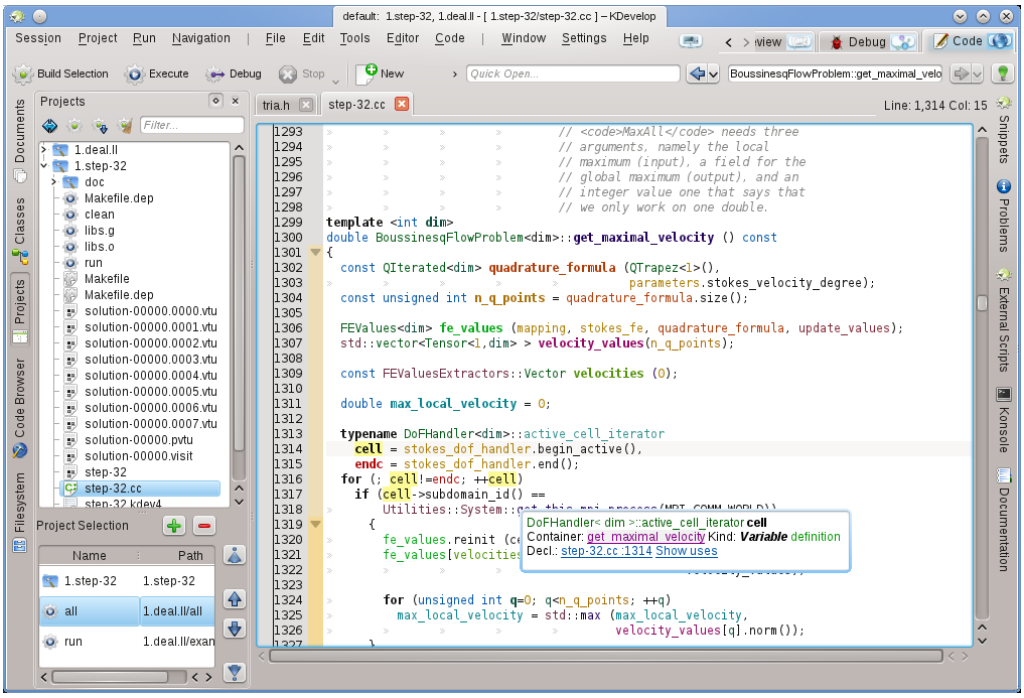
Uma lista completa das ferramentas e janelas está descrita aqui.

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

3.2 Explorar o código-fonte

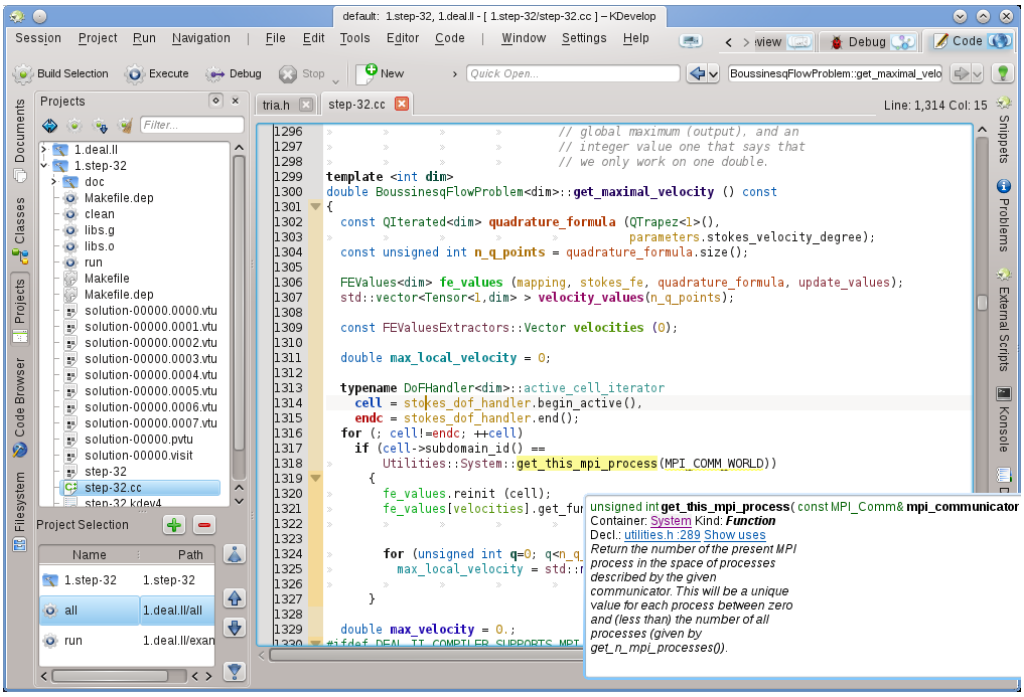
3.2.1 Informação local

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



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`, uma vez 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 atual, o símbolo sobre o qual o mouse passou não tinha documentação associada. Nesse exemplo, se o mouse tivesse passado sobre o símbolo `get_this_mpi_process`, na linha 1318, o resultado teria sido o seguinte:

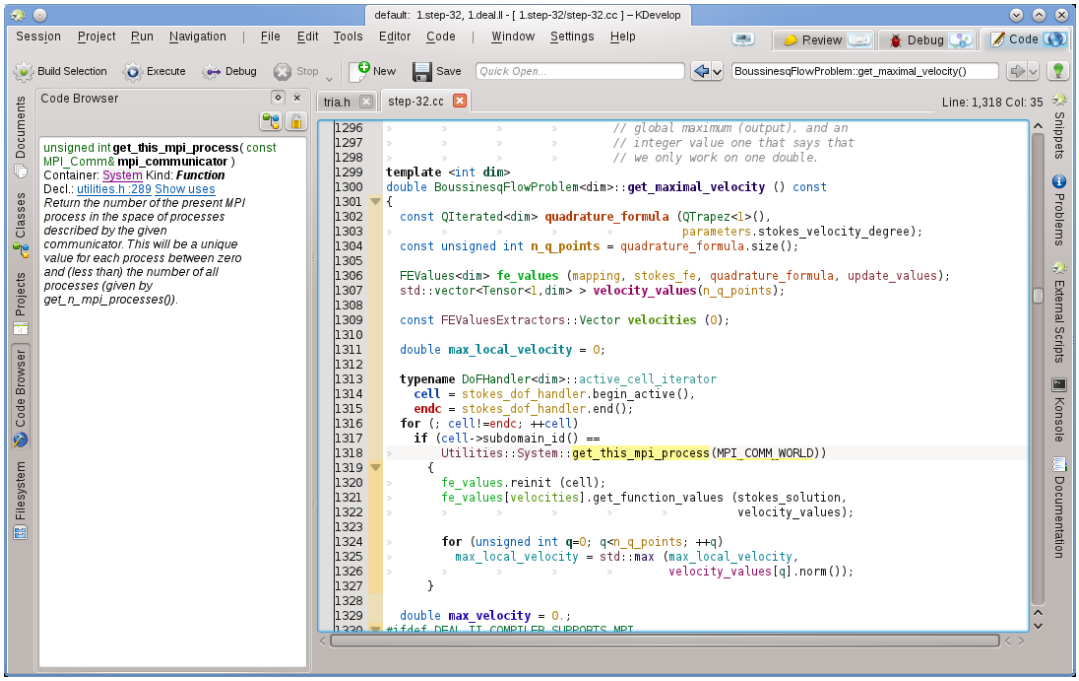


Aqui, o KDevelop cruzou a informação da declaração a partir de um arquivo completamente diferente (o `utilities.h`, que reside de fato num projeto diferente na mesma sessão), em conjunto com o comentário do 'doxygen' que acompanha a declaração nesse local.

O que torna estas dicas ainda mais úteis é o fato de serem dinâmicas: eu posso clicar no contentor para obter informações sobre o 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 clicar nas ligações azuis que irão restaurar a posição do cursor no local de declaração do símbolo (por exemplo em `utilities.h`, na linha 289) ou fornecer-lhe uma lista dos locais onde este símbolo é usado no arquivo atual ou em todos os projetos da sessão atual. 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 — isto depende se mantém pressionada a tecla **Alt** ou se passa o mouse por cima. Se quiser um local mais permanente para ela, abra a ferramenta do **Navegador de código** em uma das subjanelas. 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:



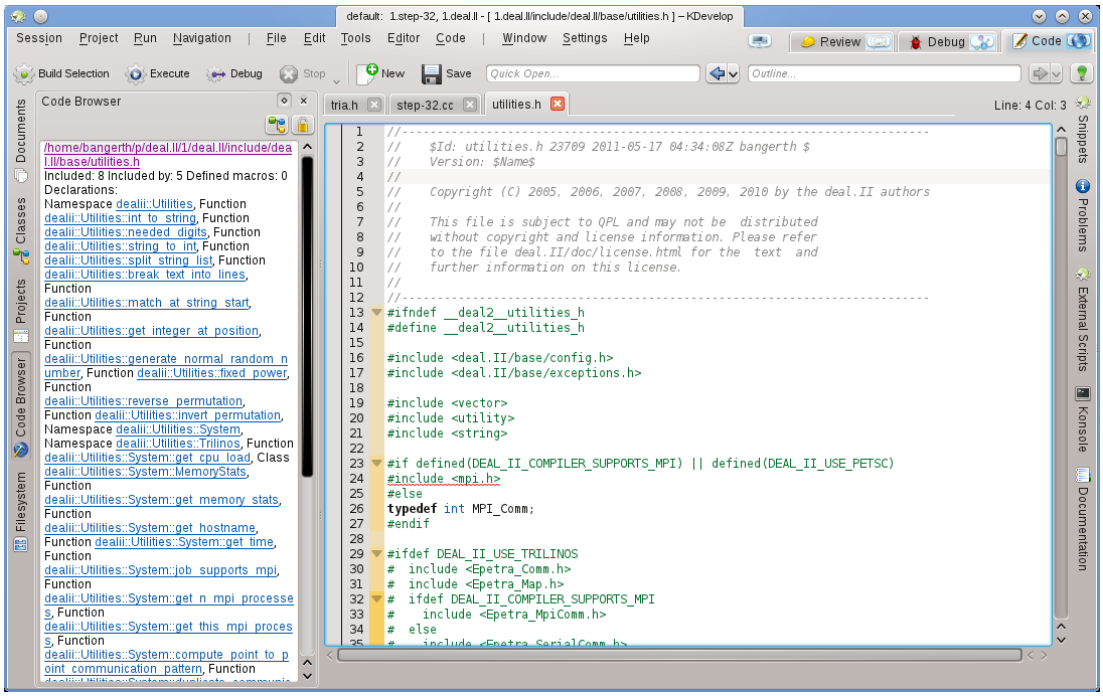
Se mover o cursor para a direita, irá mudar a informação apresentada à esquerda. Além disso, se clicar no botão **Bloquear a janela atual**, no canto superior direito, você poderá bloquear esta informação, tornando-a independente do movimento do cursor, enquanto explora a informação aí apresentada.

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 pressionada a tecla **Alt** numa lista de completamento (por exemplo ao fazer uma abertura rápida), também irá apresentar a informação de contexto do símbolo atual.

3.2.2 Informação de âmbito do arquivo

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

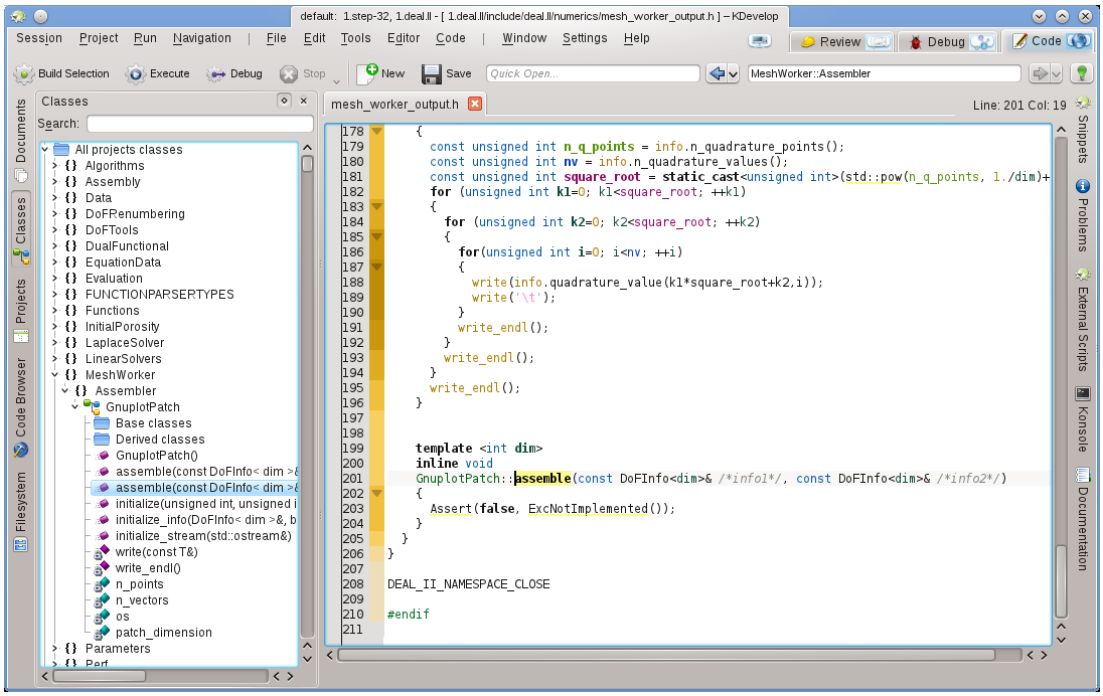


Aqui ela apresenta uma lista dos espaços de nomes, classes e funções declaradas ou definidas no arquivo atual, dando-lhe uma visão geral sobre o que se passa neste arquivo, bem como uma forma de saltar diretamente para qualquer uma destas declarações ou definições sem ter que percorrer o arquivo para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

> NOTA
> A informação apresentada a nível do arquivo 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.

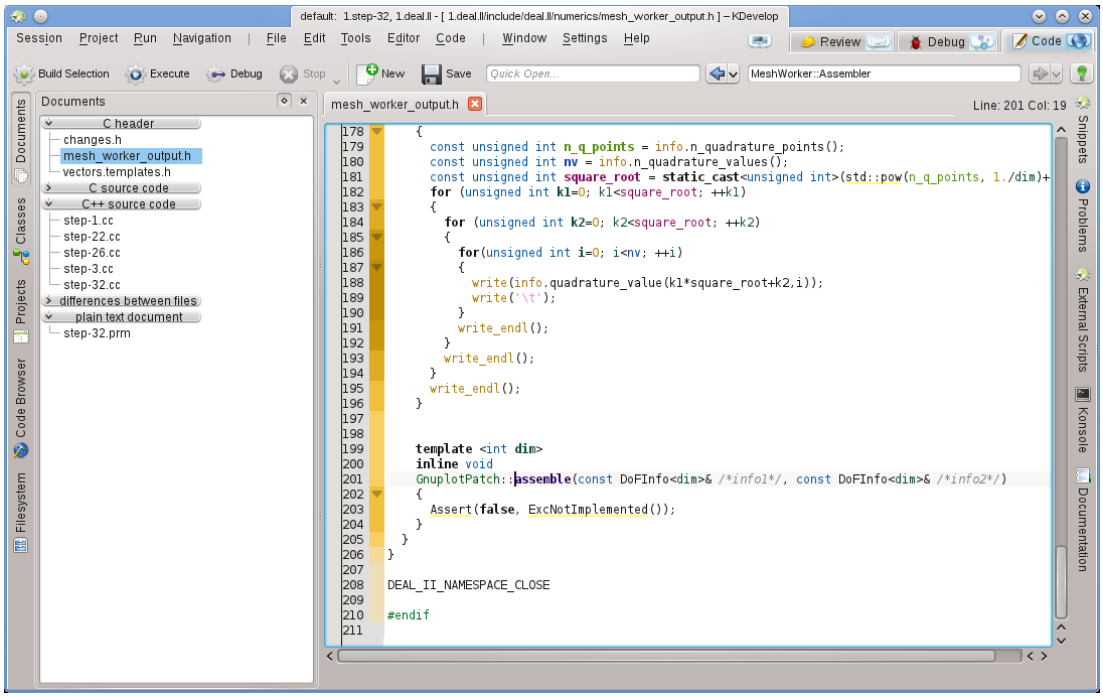
3.2.3 Informação ao nível do projeto e da sessão

Existem muitas formas de obter informações sobre um projeto inteiro (ou, de fato, sobre todos os projetos de uma sessão). Este tipo de informação é normalmente indicada 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 projetos de uma sessão, em conjunto com as funções-membro e variáveis de cada uma destas classes:



Se passar o mouse 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-clique 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 visão sobre um projeto com base nos tipos de arquivos ou outros documentos que compõem este projeto:



3.2.4 O realce do arco-íris explicado

O KDevelop usa uma variedade de cores para realçar os diferentes objetos no código-fonte. Se você souber o que as diferentes cores significam, poderá extrair rapidamente muitas informações a partir do código-fonte, bastando para isso olhar para as cores, sem precisar ler um único caractere. As regras de realce são as seguintes:

- Os objetos do tipo Classe / Estrutura, Enumerado (os valores e o tipo), as funções (globais) e os membros das classes têm cada um a sua própria cor atribuída (as classes são verdes, os enumerados são vermelho-escuro e os membros são amarelo-escuro ou violetas, sendo que as funções globais são sempre violetas).
- Todas as variáveis globais aparecem em verde-escuro.
- Os identificadores de 'typedefs' de cada tipo aparecem em verde-azulado.
- Todas as declarações e definições de objetos aparecem em negrito.
- Se um membro for acessado dentro do contexto em que é definido (classe de base ou derivada), ele aparece em amarelo, caso contrário, aparece em violeta.
- Se um membro for privado ou protegido, ele aparece com uma cor ligeiramente mais escura quando for usado.
- Para as variáveis locais de um determinado 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 objeto diferente, isto é, se for redefinido em outro nível), sendo normalmente obtida 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 ficarão com cores iguais. Estas cores do arco-íris poderão ser desativadas em separado da coloração global da janela de configuração.
- Os identificadores para os quais o KDevelop não pode determinar a declaração correspondente aparecem a branco. Isto poderá acontecer algumas vezes por instruções `#include` que estão faltando.
- Além dessa coloração, o realce de sintaxe normal do editor será aplicado, como acontece no Kate. O realce semântico do KDevelop sempre irá substituir o realce de sintaxe do editor, caso exista um conflito.

3.3 Navegar pelo código-fonte

Na seção anterior, discutimos a exploração do código-fonte, isto é, obter informações sobre os símbolos, arquivos e projetos. O passo seguinte é então navegar pelo mesmo, isto é, circular por ele todo. Existem de novo vários níveis possíveis para isso: local, dentro de um arquivo ou dentro de um projeto.

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

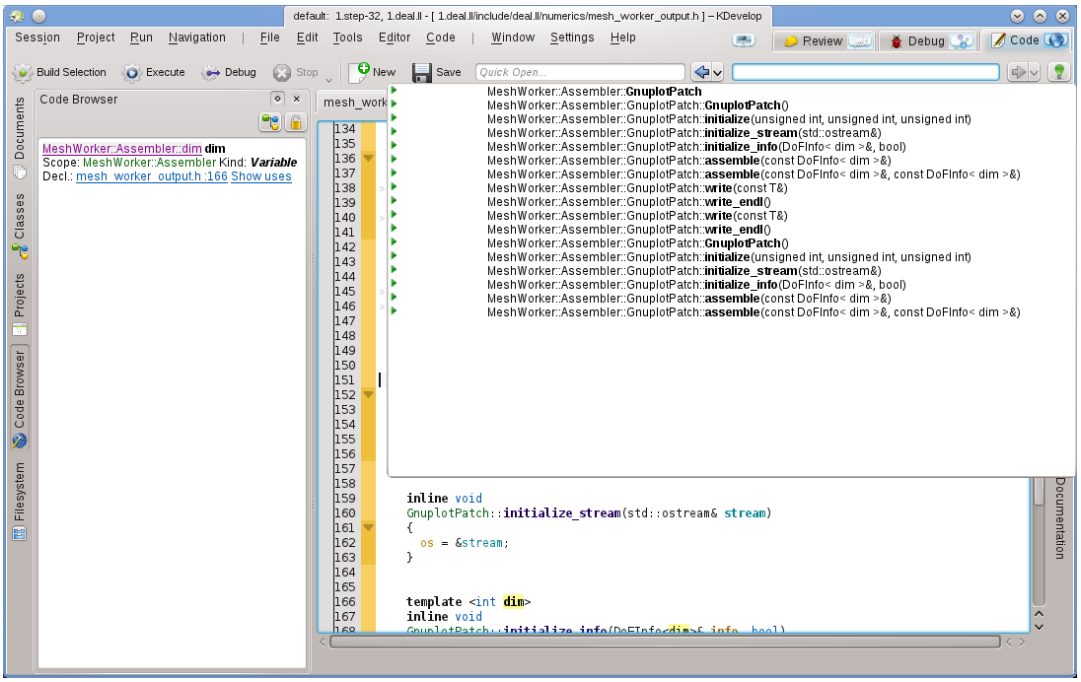
3.3.1 Navegação local

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

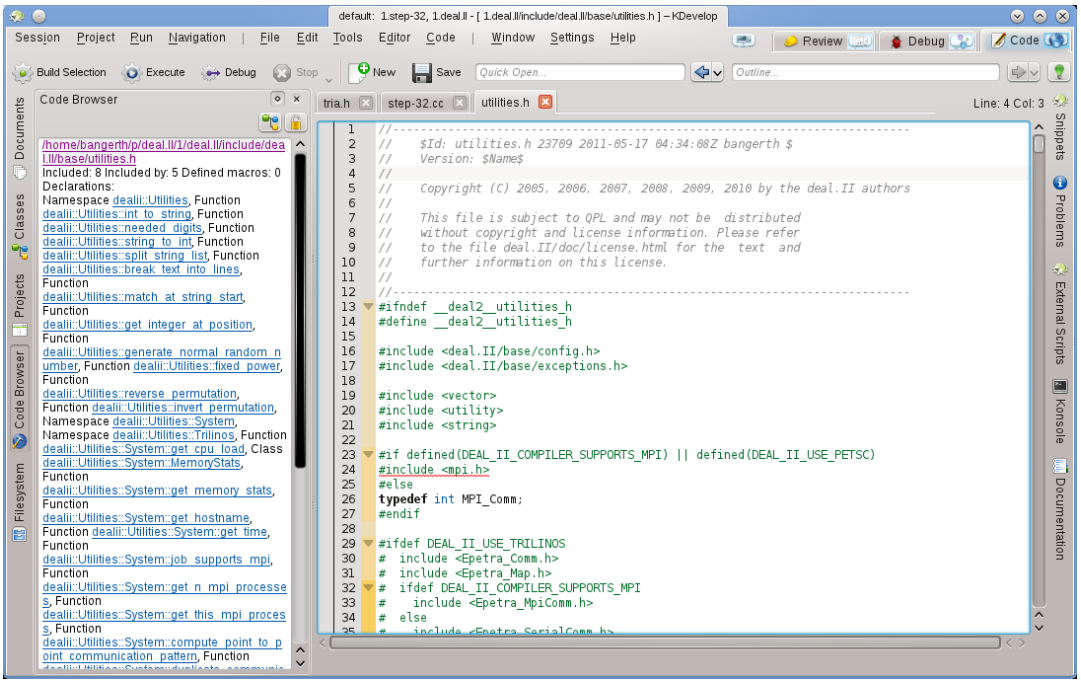
3.3.2 Navegação ao nível do arquivo e modo de contorno

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

- **Contorno**: Você poderá ter uma visão geral do que se encontra no arquivo atual, pelo menos de três formas diferentes:
  - Se clicar na área de **Contorno** no canto superior direito da janela principal, ou se pressionar **Alt-Ctrl-N**, irá abrir uma lista que apresenta todas as declarações de funções e classes:



    Você poderá então selecionar para onde desejar saltar ou — se existirem muitas — 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, uma vez que os nomes não correspondentes ao texto inserido vão sendo retirados, até que esteja pronto para selecionar uma das opções.
  - Posicionando o cursor ao nível do arquivo (isto é fora de qualquer declaração ou definição de funções ou classes) e tendo a ferramenta do **Navegador de código** aberta:



Isto também lhe dá uma ideia geral do que se passa no arquivo atual, permitindo-lhe selecionar para onde deseja ir.

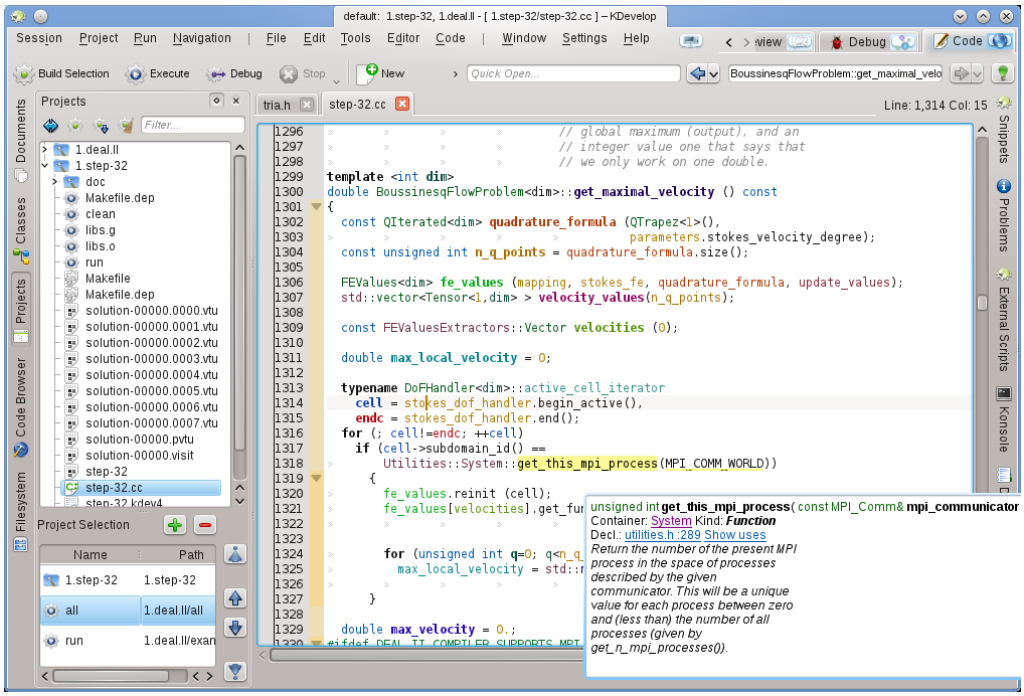
– Passar o mouse sobre o separador da página de um dos arquivos abertos também lhe dará uma visão geral do arquivo nessa página.

- Os arquivos de código estão organizados como uma lista de declarações ou definições de funções. Pressionar **Alt-Ctrl-PgUp** e **Alt-Ctrl-PgDown**, salta respectivamente para a definição de função anterior ou seguinte neste arquivo.

3.3.3 Navegação ao nível do projeto e sessão. Navegação semântica

Como foi mencionado em outros locais, o KDevelop não leva em consideração normalmente os arquivos de código individuais olhando sim para os projetos como um todo (ou para todos os projetos que façam parte da sessão atual). Em consequência, ele oferece várias possibilidades para navegar pelos projetos inteiros. Algumas destas possibilidades são derivadas do que já foi discutido na seção 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 necessite processar os projetos por inteiro e interligar os dados. A lista a seguir mostra-lhe algumas formas de navegar pelo código-fonte que esteja espalhado por uma grande quantidade de arquivos:

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



Se clicar 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 arquivo e colocando o cursor na posição correspondente. Você poderá obter um efeito semelhante se usar a ferramenta do **Navegador de código**, que também foi descrita anteriormente.

- Um modo rápido de saltar para a declaração de um símbolo sem ter que clicar nos links da dica é habilitar temporariamente **Modo de navegação no código** segurando a tecla **Alt** ou **Ctrl**. Neste modo, é possível clicar em qualquer símbolo no editor para saltar para sua declaração.

- **Abertura rápida**: Uma forma bastante poderosa de saltar para outros arquivos ou locais é usar os vários métodos de *abertura rápida* no KDevelop. Existem quatro versões destes métodos:

  – **Abrir rapidamente a classe** (**Navegar** → **Abrir rapidamente a classe** ou **Alt-Ctrl-C**): Você obterá uma lista com todas as classes nesta sessão. Comece a digitar (uma parte de) o nome de uma classe para que a lista vá se reduzindo para mostrar apenas as que corresponderem ao texto escrito até agora. Se a lista for pequena o suficiente, selecione 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**): Você obterá uma lista com todas as funções 'membros' que fazem parte dos projetos na sessão atual, podendo selecionar, a partir desta lista 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 arquivo** (**Navegar** → **Abrir rapidamente o arquivo** ou **Alt-Ctrl-O**): Você obterá uma lista com todos os arquivos que fazem parte dos projetos na sessão atual, onde poderá escolher o arquivo em questão da mesma forma que foi descrita acima.

  – **Abertura rápida universal** (**Navegar** → **Abertura rápida** ou **Alt-Ctrl-Q**): Se você 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 arquivos, funções, classes e outros itens que possa selecionar.

- **Ir para a declaração/definição**: Ao implementar uma função-membro, normalmente uma pessoa precisa 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 atualizar a documentação. Para fazer isso, coloque o cursor sobre o nome da função e selecione a opção **Navegação** → **Ir para a declaração** (ou pressione **Ctrl-.**) para ir para o local onde está declarada a função. Existem várias formas de voltar ao local original:

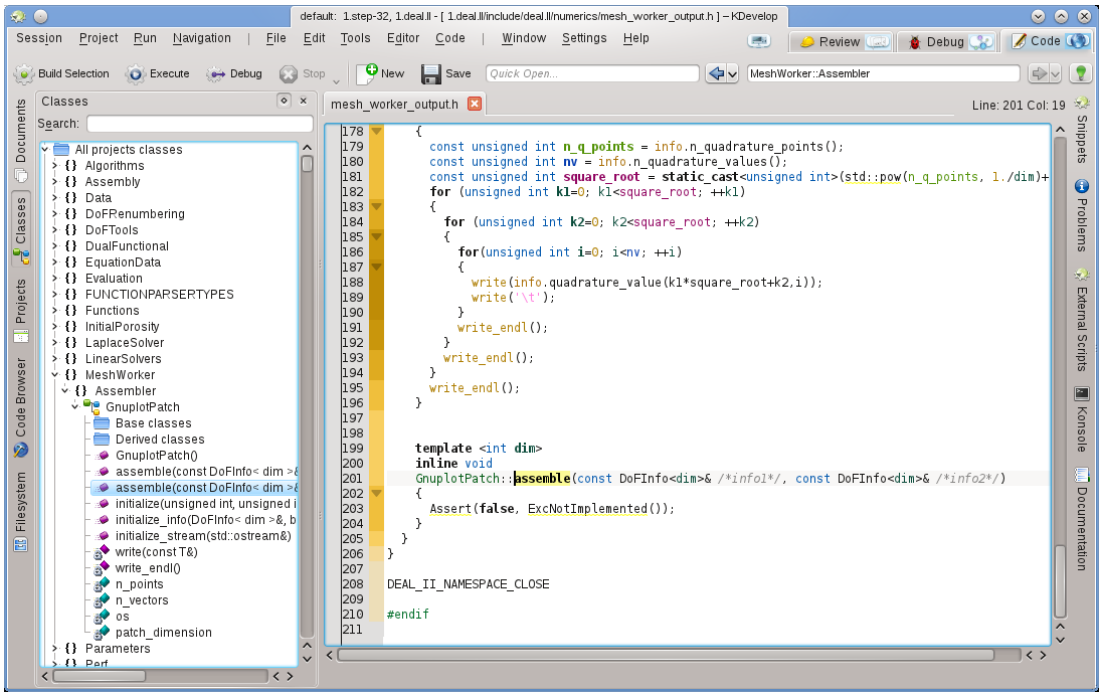
  - Selecionando a opção **Navegação** → **Ir para a definição** (ou pressionando **Ctrl-,**).
  - Selecionando a opção **Navegação** → **Contexto visitado anterior** (ou pressionando **Meta-Esquerda**), como descrito abaixo.

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 implementando no 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.

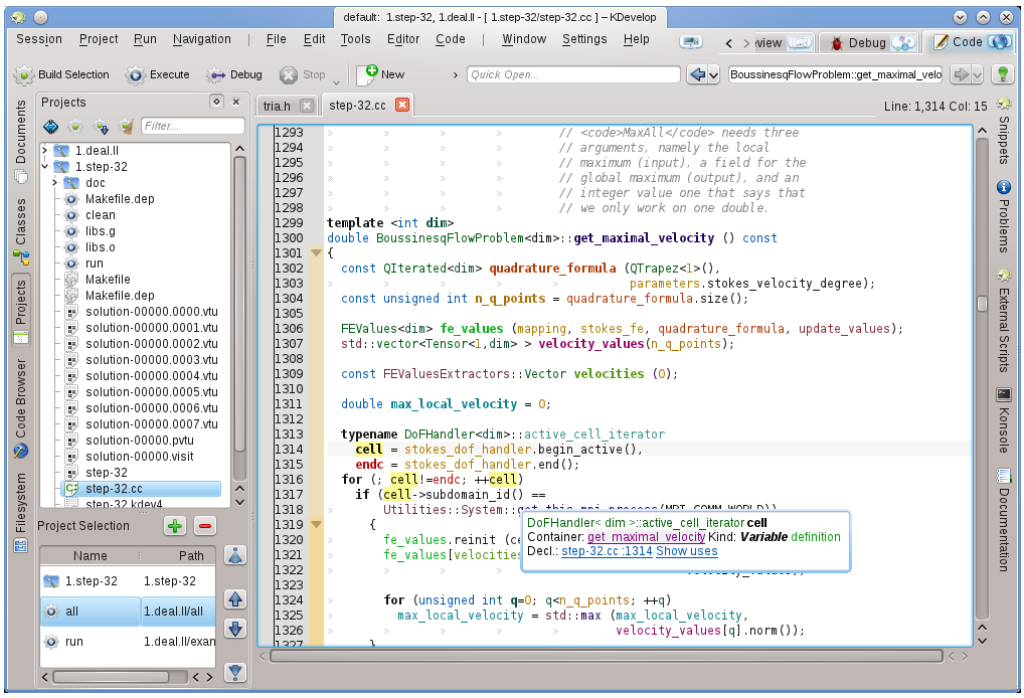
- **Alternar entre a declaração/definição**: No exemplo acima, para ir para o local da declaração da função atual, você terá primeiro que colocar o cursor sobre o nome da função. Para evitar este passo, poderá selecionar a opção **Navegação** → **Alternar entre a definição/declaração** (ou pressione **Shift-Ctrl-C**) para ir para a declaração da função onde se encontra o cursor no momento. Se selecionar 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 selecionar a opção **Navegação** → **Uso seguinte** (ou pressionar **Meta-Shift-Direita**) irá para a utilização seguinte desta variável no código. (Lembre-se de que isto não pesquisa apenas pela ocorrência seguinte da variável mas também considera as variáveis com o mesmo nome, mas em âmbitos diferentes). O mesmo resulta para a utilização dos nomes das funções. Se selecionar **Navegação** → **Uso anterior** (ou pressionar **Meta-Shift-Esquerda**), irá para a utilização anterior de um determinado símbolo.

NOTA
Para ver uma lista com todas as utilizações de um nome, coloque o cursor sobre ele e abra a ferramenta do **Navegador de código** ou pressione e mantenha pressionada a tecla **Alt**. Isto é explicado com mais detalhes na seção 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, exceto que, em vez de páginas Web, você visita os *contextos*. Um contexto é a localização atual do cursor e o usuário poderá alterá-la se navegar para fora dela, usando tudo menos os comandos de cursores — por exemplo, se clicar 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 projetos da sessão atual, permitindo-lhe expandi-la para ver as funções e variáveis membros de cada uma destas classes:



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



Mais uma vez, se fizer duplo-clique sobre um arquivo, irá abri-lo.

3.4 Escrever código-fonte

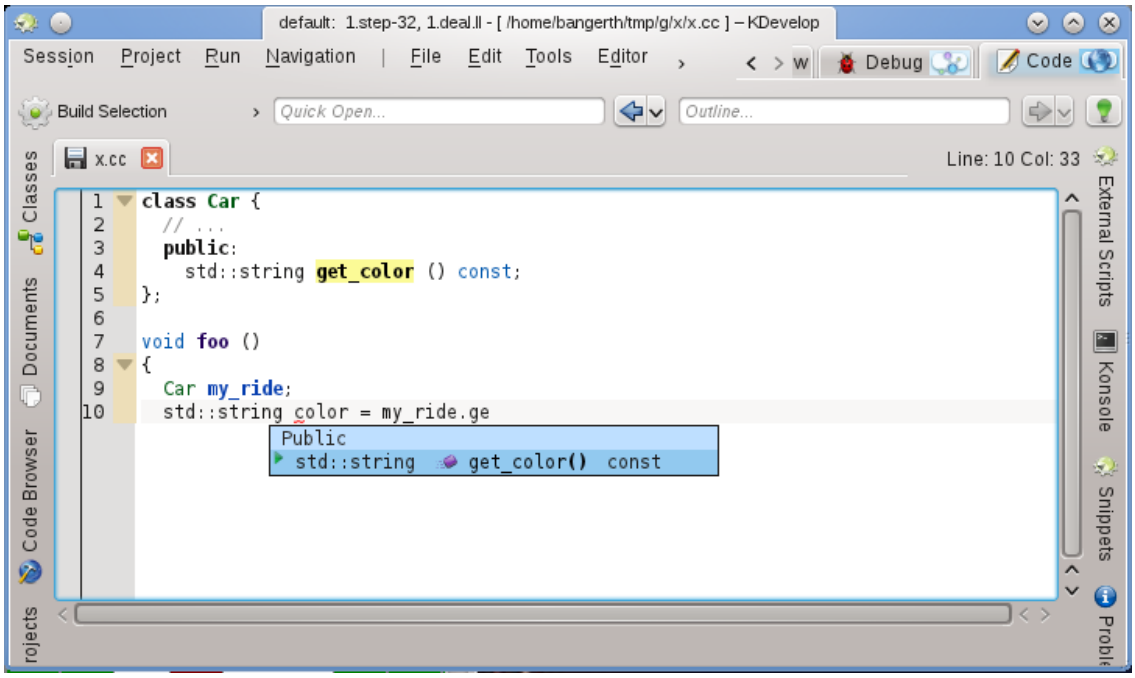
Uma vez que o KDevelop compreende o código-fonte dos seus projetos, ele poderá ajudá-lo a escrever mais código. Os pontos a seguir descrevem algumas das formas como isso pode ser feito.

3.4.1 Complementação automática

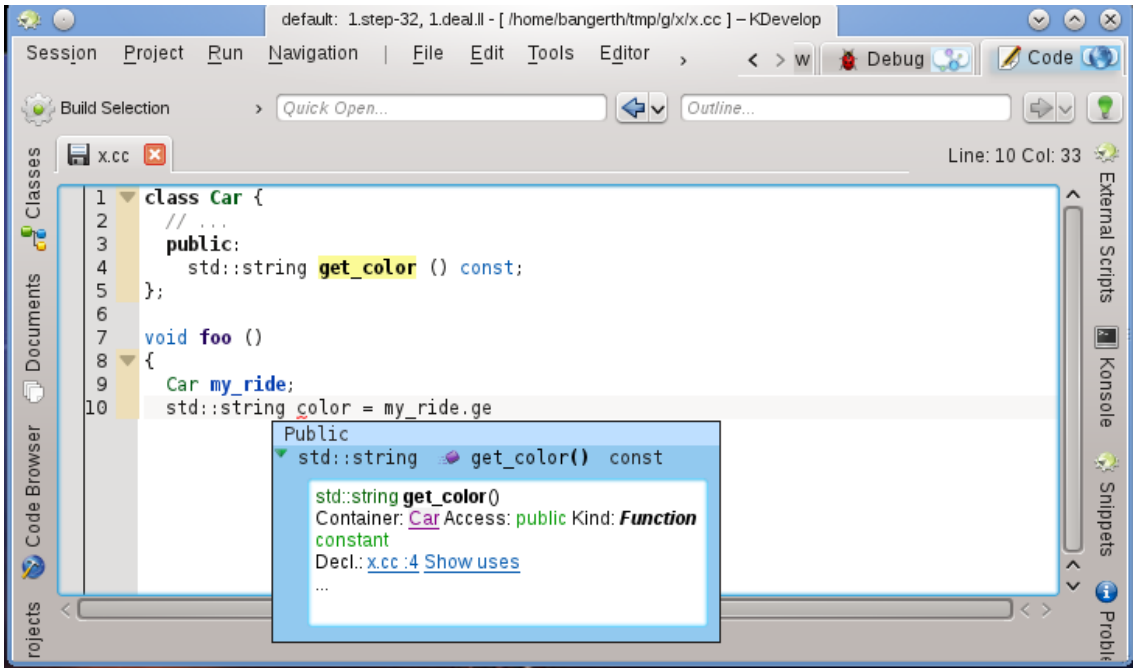
Provavelmente a funcionalidade mais útil de todas na escrita de código novo é a complementação automática. Considere, por exemplo, o seguinte pedaço de código:

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

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



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



A complementação automática poderá poupar bastante escrita se o seu projeto usar nomes de variáveis e funções compridos; além disso, evita os enganos nos nomes (e os erros de compilação daí resultantes) e torna muito mais simples recordar os nomes exatos das funções; por exemplo, se todos os seus métodos de leitura começarem por `get_` (ler_), então a funcionalidade de complementação 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 correta. Lembre-se que, para a complementação automática funcionar, nem a declaração da classe `Carro` nem da variável `meu_carro` terão que estar no mesmo arquivo onde está escrevendo o código no momento. O KDevelop simplesmente tem que saber onde estão ligadas estas classes e variáveis, isto é os arquivos aos quais é necessário ter estas ligações feitas terão que fazer parte do projeto onde está trabalhando.

NOTA

O KDevelop nem sempre sabe quando deverá auxiliá-lo a completar o código. Se a dica de complementação automática não abrir automaticamente, pressione **Ctrl**-**Espaço** para abrir uma lista de complementações manualmente. De um modo geral, para a complementação automática funcionar, o KDevelop precisa de processar os seus arquivos de código. Isto acontece em segundo plano para todos os arquivos que fizerem parte dos projetos da sessão atual, após iniciar o KDevelop, assim como após o usuário terminar de escrever durante uma fração de segundo (o atraso pode ser configurado).

NOTA

O KDevelop só processa arquivos que ele considere como sendo código-fonte, de acordo com o tipo MIME do arquivo. Este tipo não está definido até a primeira vez em que um arquivo é salvo; em consequência, ao criar um arquivo novo e ao começar a escrever código, ele não ativará o processamento da complementação automática até que seja salvo pela primeira vez.

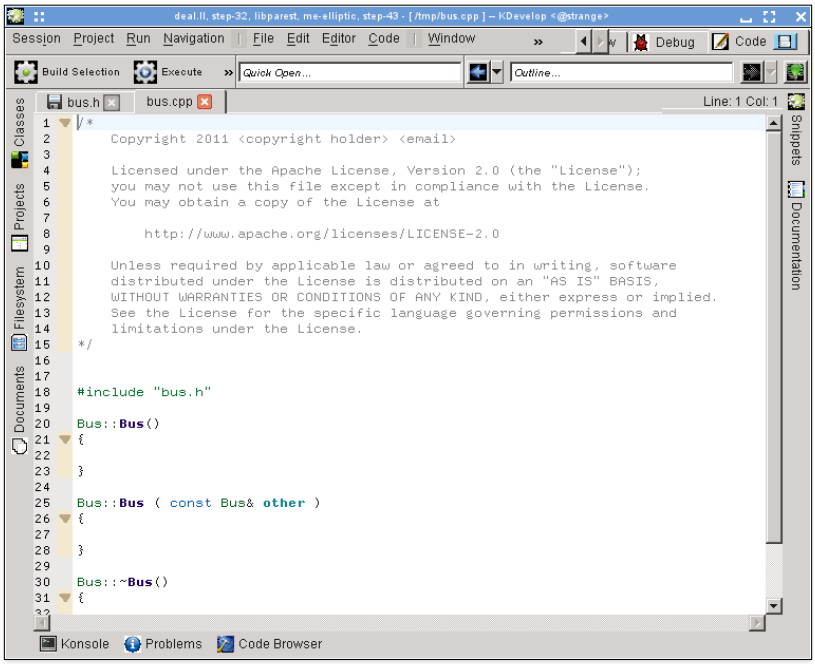
NOTA
Como na nota anterior, para a complementação automática funcionar, o KDevelop terá que conseguir descobrir as declarações nos arquivos de inclusão. Para isso, ele procura num conjunto de locais predefinidos. Se não encontrar automaticamente um arquivo de inclusão, irá sublinhar o nome de um arquivo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

NOTA
A configuração da complementação automática é discutida nesta seção deste manual.

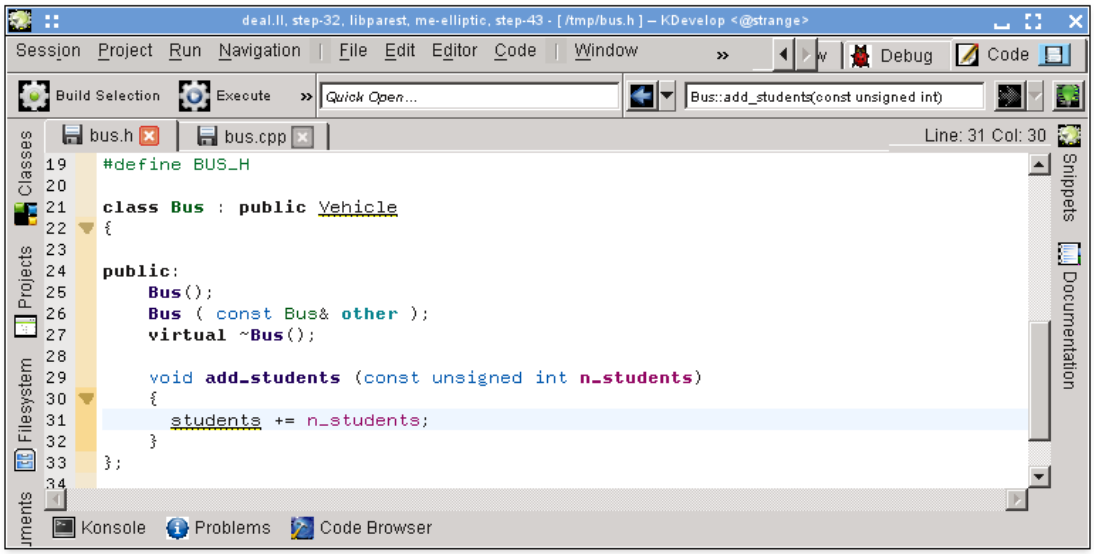
3.4.2 Adicionar classes novas e implementar as funções-membro

O KDevelop possui um assistente para adicionar novas classes. O procedimento é descrito em Criando uma nova classe. Uma classe C++ simples pode ser criada selecionando o modelo C++ Básico a partir da categoria `Classe`. No assistente, nós podemos selecionar algumas funções-membro predefinidas, por exemplo um construtor vazio, um construtor de cópia e um destrutor.

Após completar o assistente, os novos arquivos são criados e abertos no editor. O arquivo de inclusão já contém guardas de inclusão e a classe nova tem todas as funções-membro que selecionamos. 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 **onibus.cpp** onde se encontra já o esqueleto das funções:

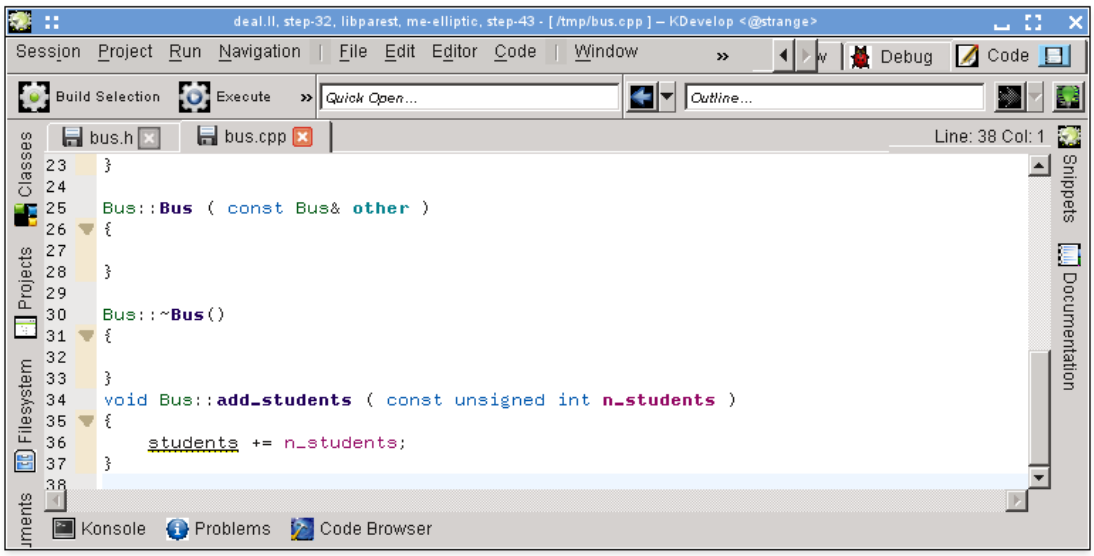


Para adicionar novas funções-membro, volte ao arquivo **onibus.h** e adicione o nome de uma função. Por exemplo, adicione o seguinte:



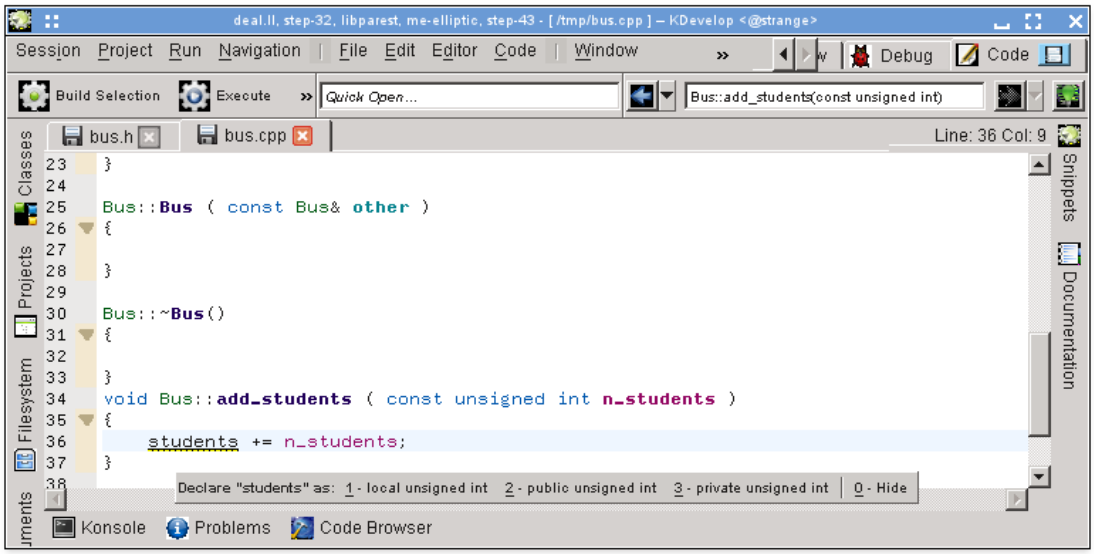

```
19  #define BUS_H
20
21  class Bus : public Vehicle
22  {
23
24  public:
25      Bus();
26      Bus ( const Bus& other );
27      virtual ~Bus();
28
29      void add_students (const unsigned int n_students)
30      {
31        students += n_students;
32      }
33  };
34
```

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

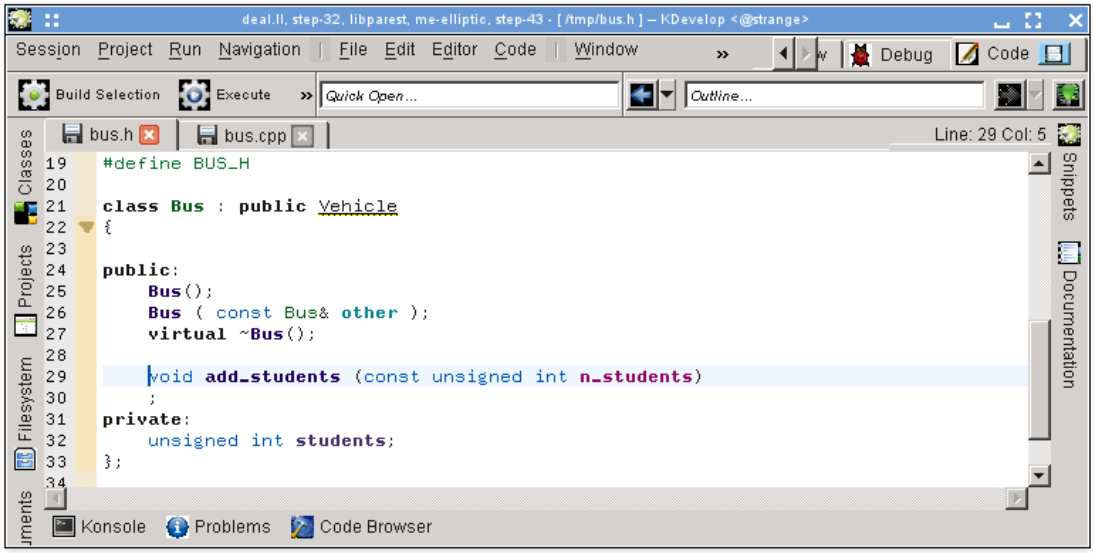



```
23  }
24
25  Bus::Bus ( const Bus& other )
26  {
27
28  }
29
30  Bus::~Bus()
31  {
32
33  }
34  void Bus::add_students ( const unsigned int n_students )
35  {
36      students += n_students;
37  }
38
```

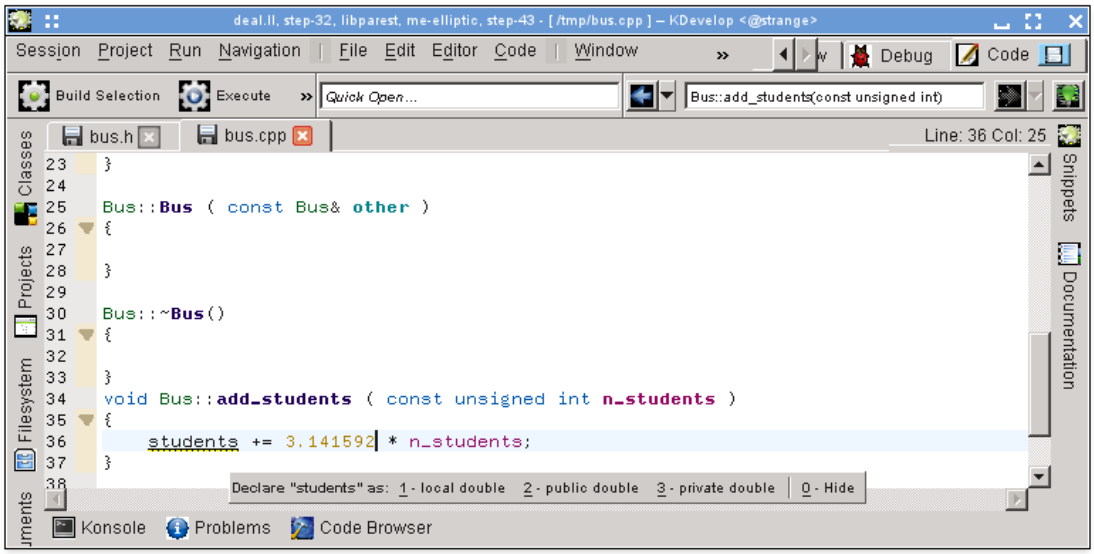
Repare que eu acabei de digitar e desejava inferir que a variável `estudantes` deveria ser provavelmente uma variável-membro da classe `Ônibus`, mas esta ainda não foi adicionada. Repare também como o KDevelop a sublinha para realçar que ainda não sabe nada sobre a variável. Contudo, este problema pode ser resolvido: se clicar no nome da variável, irá aparecer a seguinte dica:



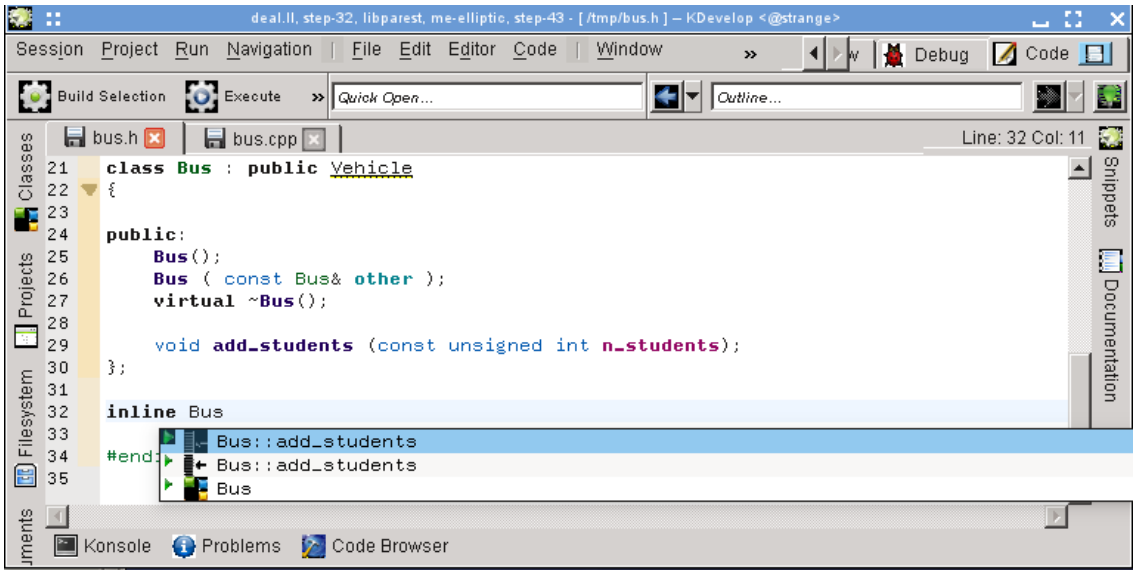
(O mesmo pode ser obtido se clicar com o botão direito sobre o mesmo e selecionar **Resolver: Declarar como**.), podendo selecionar '3 - private unsigned int' (com o mouse, ou pressionando **Alt-3**) e ver como irá aparecer no arquivo de inclusão:



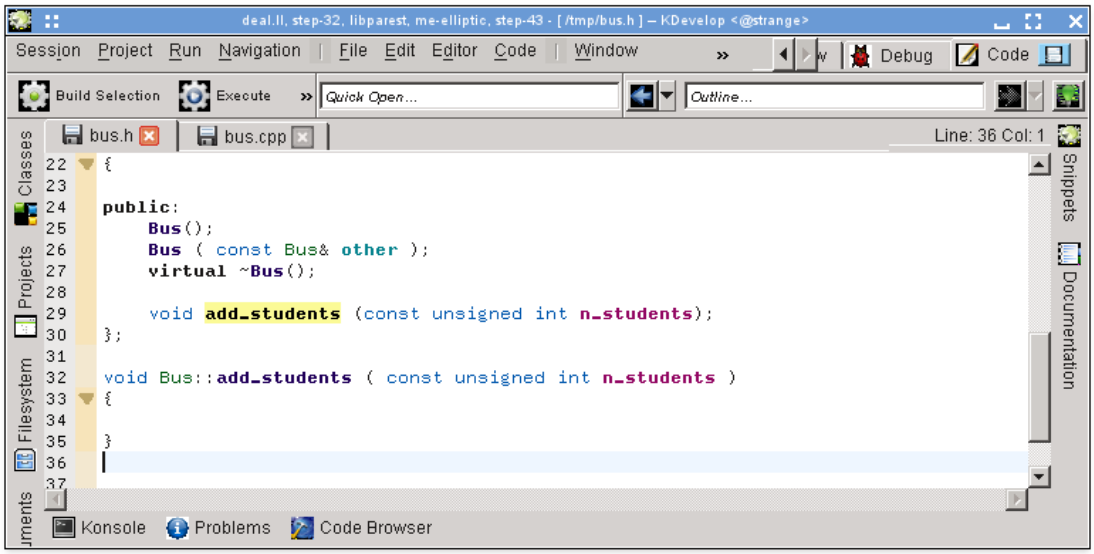
É importante referir que o KDevelop extrai o tipo da variável a declarar a partir da expressão usada para a inicializar. Por exemplo, se tivéssemos escrito a soma na seguinte forma, ainda que dúbia, ele teria sugerido que a variável fosse declarada como `double`:



Como ponto final: O método que usa o **Código** → **Mover para o código** nem sempre insere a nova função-membro onde se deseja. Por exemplo, você poderá querer marcá-la como `inline` e colocá-la no fundo do arquivo 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 complementações possíveis do que possa aparecer aqui. Se selecionar 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 opções na ferramenta de complementação automática, irá mostrar a assinatura correta, mas infelizmente apaga o marcador `inline` já escrito. Isto foi comunicado como sendo o Erro 274245 do KDevelop.

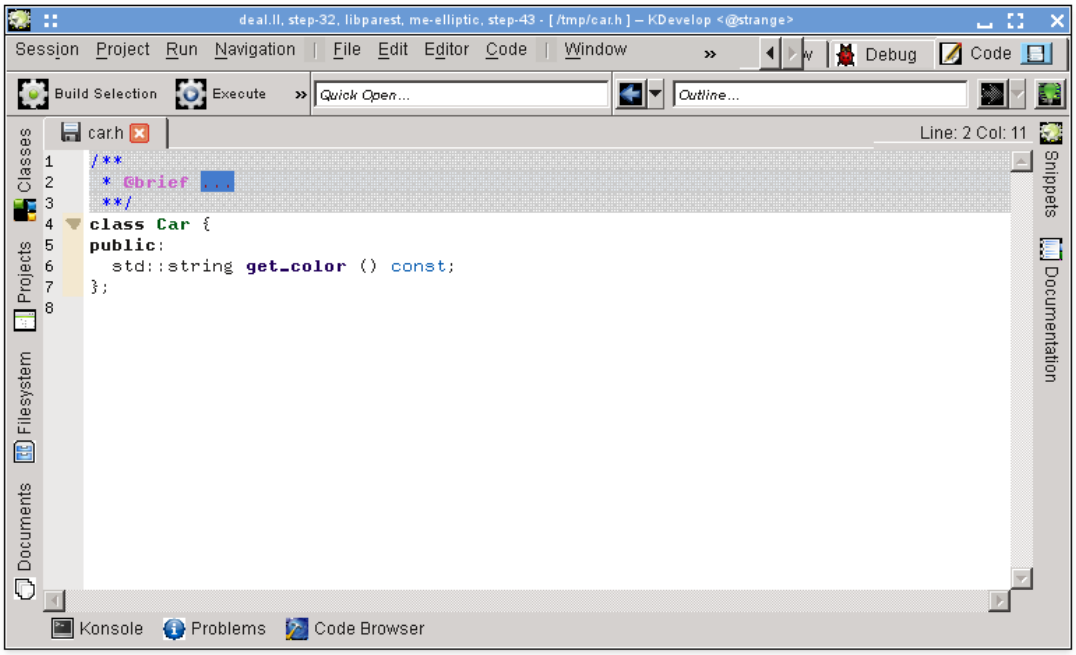
3.4.3 Documentar as declarações

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 objetivo de explicar o seu objetivo, 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 fato para formatar os comentários para que possam ser extraídos e apresentados em páginas Web navegáveis.

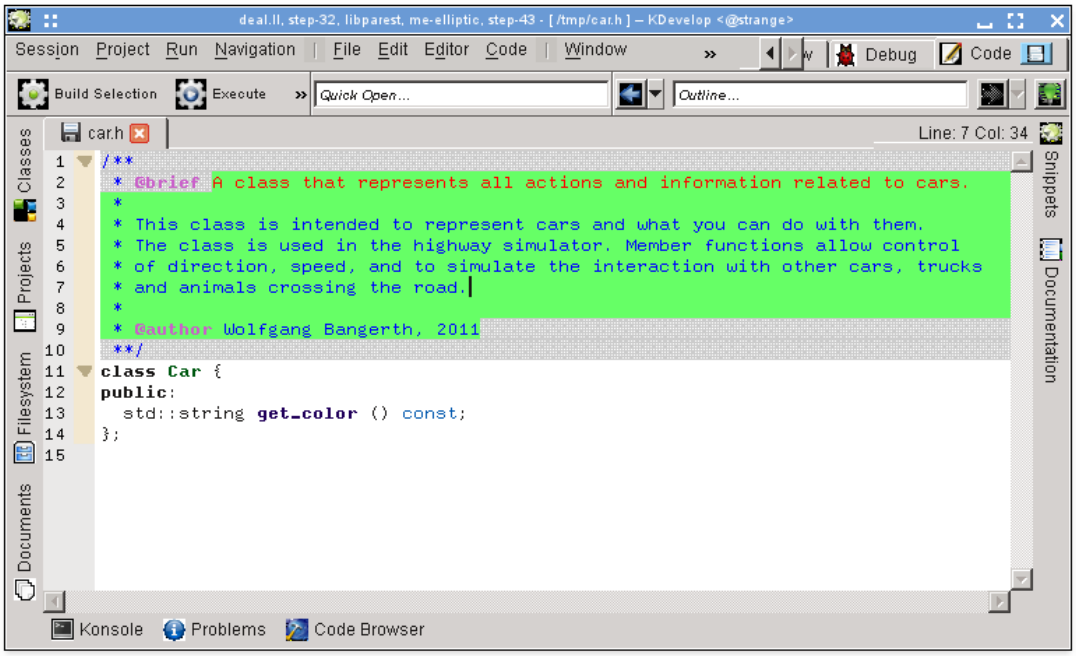
O KDevelop suporta este estilo de comentários, contendo um atalho para gerar a estrutura de comentários que documentam uma classe ou função-membro. Por exemplo, assumindo que já tenha escrito este código:

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

Você desejará agora adicionar a documentação tanto à classe como à função-membro. Para isso, mova o cursor para a primeira linha e selecione **Código** → **Documentar a declaração** ou pressione **Alt-Shift-D**. O KDevelop irá responder com o seguinte:

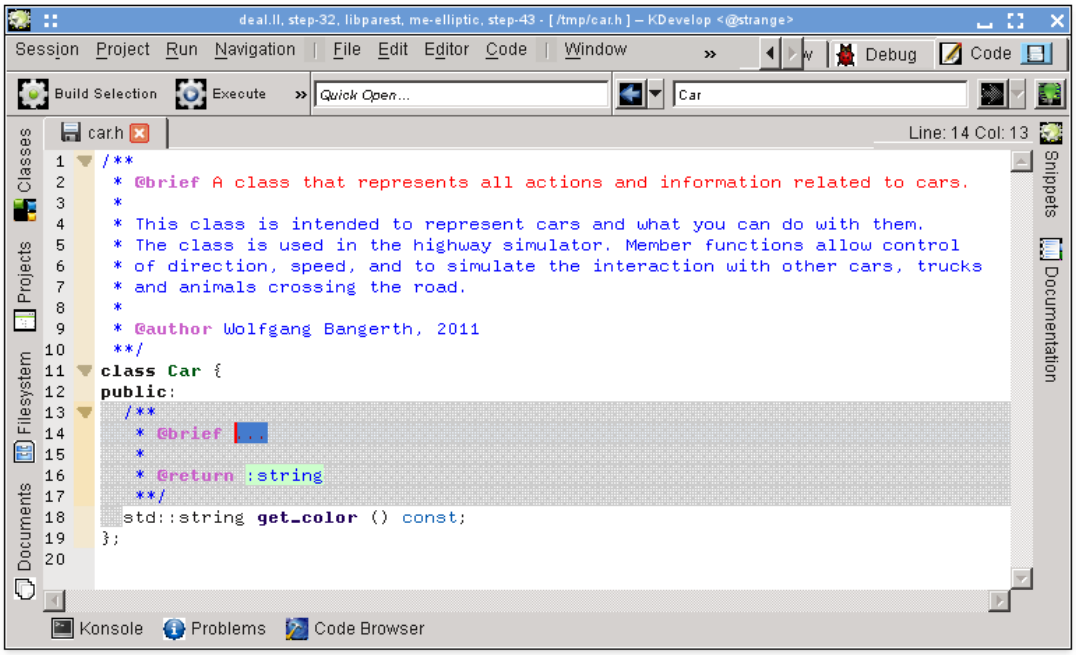


O cursor já se encontra na área em cinza para você preencher a breve descrição (depois da palavra-chave do 'doxygen' `@brief`) desta classe. Você poderá então continuar a adicionar a documentação a este comentário, dando uma descrição mais detalhada sobre o que a classe faz:

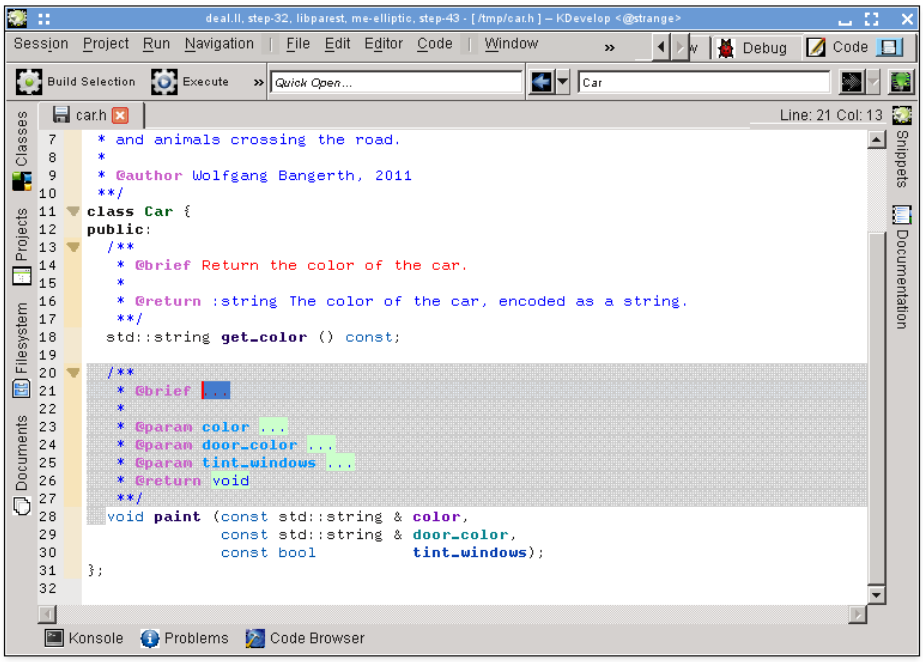


Enquanto o editor estiver dentro do comentário, o texto do mesmo fica realçado em verde (o realce desaparece assim que sair do comentário). Quando for para o fim de uma linha, pressione **Enter** para que o KDevelop inicie uma nova linha começando com um asterisco e coloca o cursor com um caractere de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

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



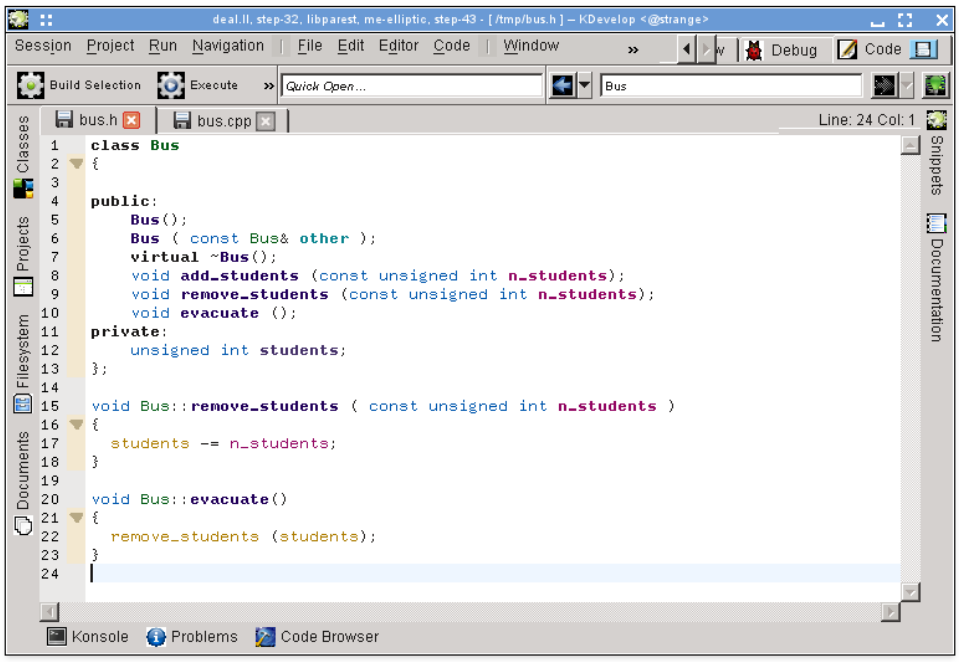
Mais uma vez, o KDevelop irá gerar automaticamente o esqueleto de um comentário, incluindo a documentação da função em si, assim como o tipo devolvido por esta. No caso atual, o nome da função é bastante intuitivo, mas muitas das vezes os argumentos da função poderão não ser e, como tal, deverão ser 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 Renomear as variáveis, funções e classes

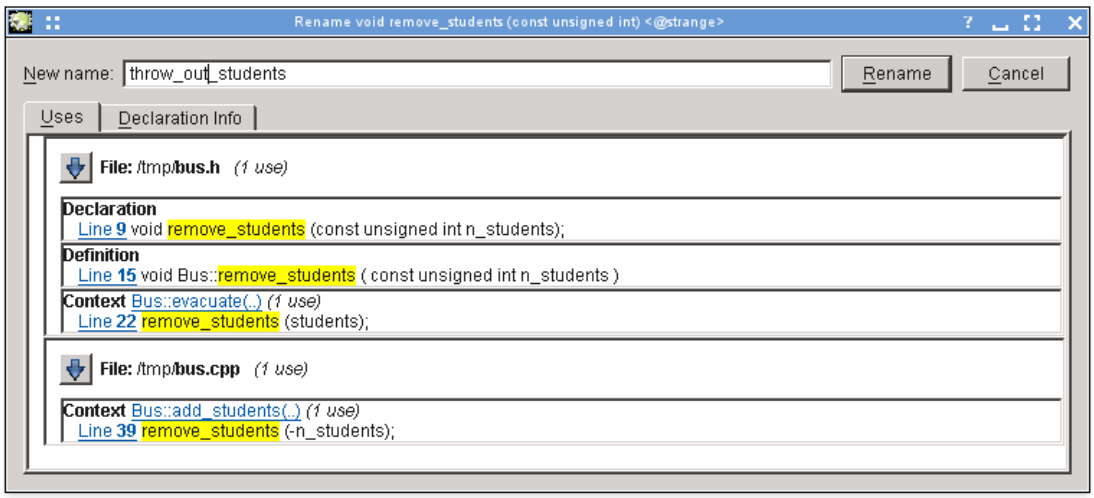
Algumas vezes, alguém poderá querer renomear uma função, classe ou variável. Por exemplo, imagine que nós já temos o seguinte:



Iremos então concluir que estamos insatisfeitos com o nome `remover_estudantes` e que se deveria chamar por exemplo `soltar_estudantes`. Poderíamos fazer uma pesquisa-substituição por esse nome, mas isso tem duas desvantagens:

- A função pode ser usada em mais de um arquivo.
- Realmente só queremos mudar o nome desta função e não tocar nas funções que possam ter o mesmo nome mas que estejam declaradas em outras 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 selecionar **Código** → **Renomear a declaração** (ou se clicar com o botão direito no nome e selecionar a opção **Renomear Onibus::remover_estudantes**). Isto irá invocar uma janela onde poderá indicar o novo nome da função e onde poderá ver todos os locais onde ela é usada:

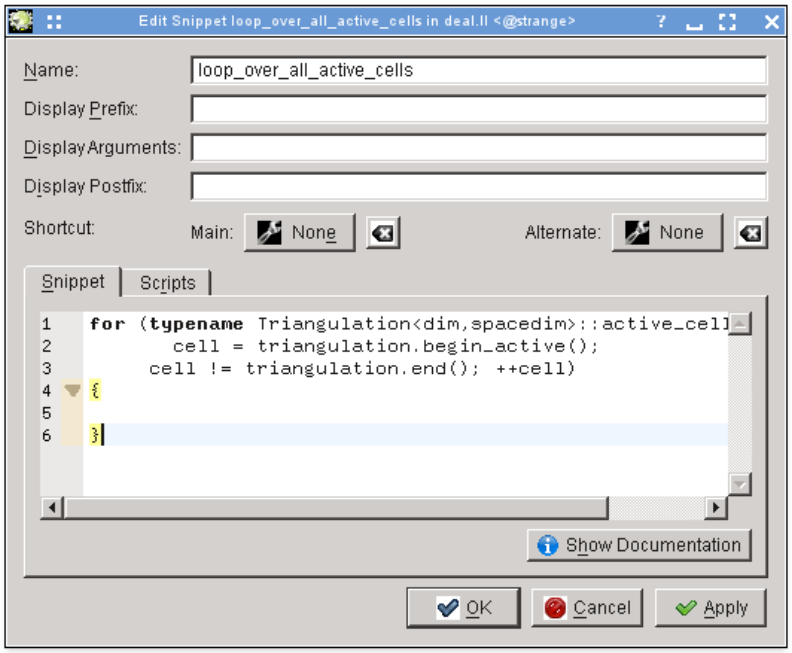


3.4.5 Trechos de código

A maioria dos projetos possuem pedaços de código que uma pessoa terá que 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 usuários, verificar se os dados do usuário são válidos e, caso contrário, mostrar uma mensagem de erro; no projeto 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 ...
```

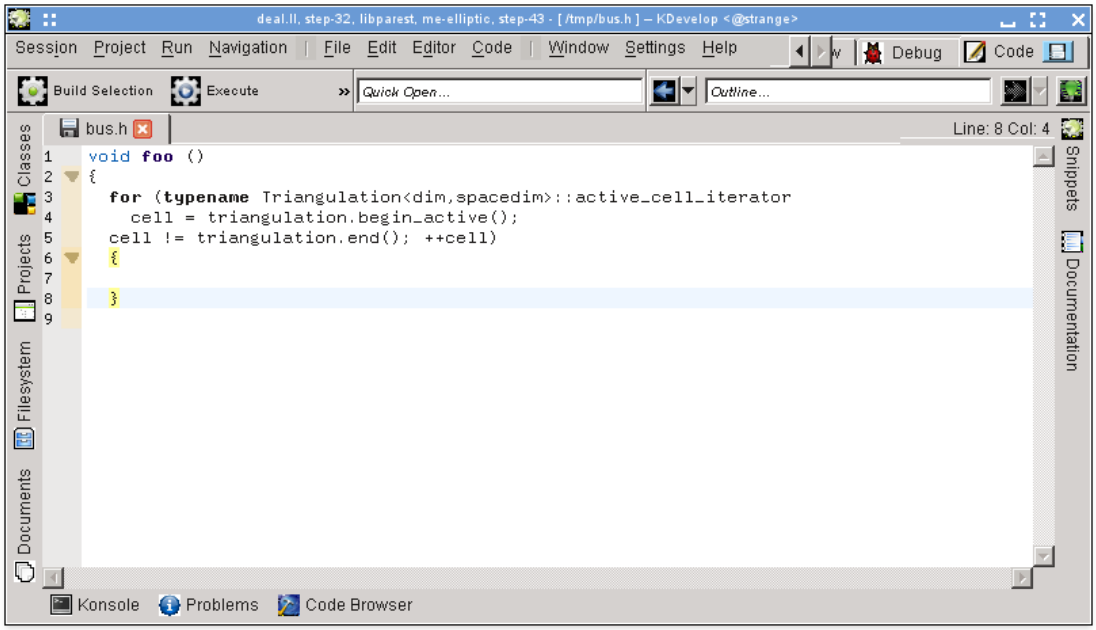
Em vez de escrever este tipo de texto repetidamente (com todos os erros associados que isso possa introduzir), a ferramenta de **Trechos** do KDevelop poderá ajudá-lo aqui. Para isso, abra a área de ferramentas (veja em Ferramentas e janelas se o botão correspondente não existir já no entorno da sua janela). Depois clique no botão 'Adicionar um repositório' (um nome ligeiramente confuso — ele permite-lhe criar uma coleção de trechos com um determinado nome para os arquivos de código de um determinado tipo, por exemplo código em C++) e crie um repositório vazio. Depois, clique em  para adicionar um trecho, obtendo uma janela como a seguinte:



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

Para usar o trecho assim definido, quando estiver editando o código, basta escrever o nome do trecho como o faria com qualquer função ou variável. Este nome ficará disponível na complementação automática — o que significa que não haverá qualquer problema em usar nomes compridos e descritivos para um trecho, como o descrito acima — e quando aceitar a dica de sugestão da

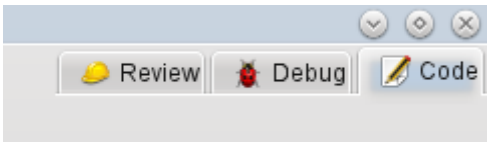
complementação automática (por exemplo, pressionando apenas em **Enter**), a parte já introduzida do nome do trecho será substituída pela expansão completa do trecho e será devidamente indentada:



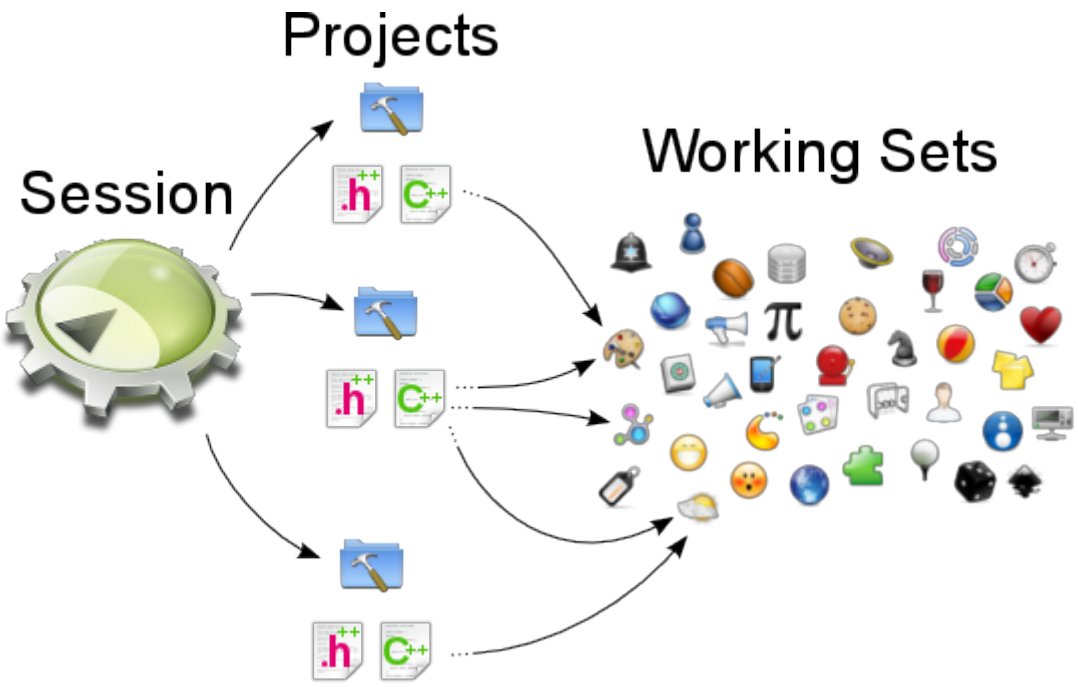
Lembre-se que, para isto funcionar, a ferramenta de **Trechos** não precisa de estar aberta ou visível: só irá precisar da ferramenta para definir trechos novos. Uma alternativa, embora menos conveniente, para expandir um trecho é simplesmente clicar nele na área de ferramentas respectiva.

> NOTA
> Os trechos 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 Trechos.

3.5 Modos e conjuntos de trabalho

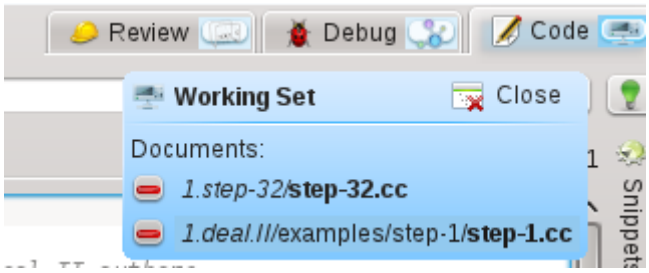


Se você tiver chegado até aqui, dê uma olhada na parte superior direita da janela principal do KDevelop. Como aparece na imagem, irá reparar que existem três **modos** possíveis para o KDevelop: **Código** (o modo que discutimos no capítulo anterior ao lidar com o código-fonte), **Depuração** (veja como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

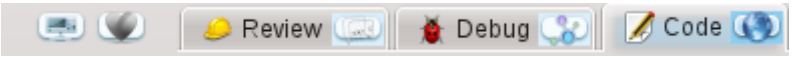


Cada modo tem o seu próprio conjunto de ferramentas espalhadas em torno da janela principal, e cada modo também tem um *conjunto de trabalho* dos arquivos e documentos abertos no momento. Além disso, cada um destes conjuntos de trabalho está associado com uma sessão atual, isto é temos a mesma relação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

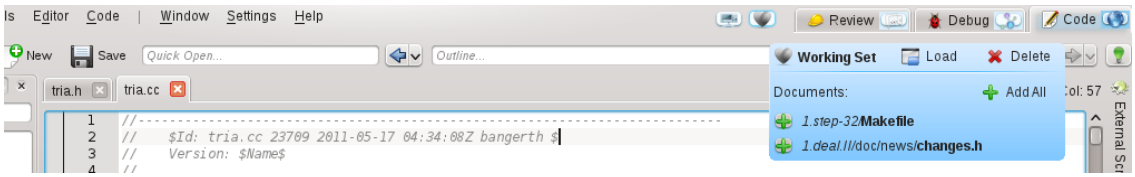
Se você abrir o KDevelop da primeira vez, o conjunto de trabalho está vazio — não existem arquivos abertos. Porém, à medida que você abre os arquivos para edição (ou depuração ou revisão nos outros modos), o seu conjunto de trabalho vai crescendo. O fato de o seu conjunto de trabalho não estar vazio é indicado através de um símbolo na página, como demonstrado abaixo. Irá reparar que, sempre que fechar o KDevelop e reiniciá-lo, o conjunto de trabalho é salvo e restaurado, isto é irá obter o mesmo conjunto de arquivos abertos.



Se você passar o seu mouse sobre o símbolo do conjunto de trabalho, irá obter uma dica que lhe mostra os arquivos que estão abertos no momento neste conjunto de trabalho (aqui: os arquivos `passo-32.cc` e `passo-1.cc`). Se clicar no sinal de somar vermelho, irá fechar a página do arquivo correspondente. Talvez ainda mais importante, se clicar no botão com o nome correspondente, irá **fechar** todo o conjunto de trabalho de uma vez (isto é fechar todos os arquivos abertos no momento). O ponto importante sobre o fechamento do conjunto de trabalho, contudo, é que não só fecham todos os arquivos, como também salva o conjunto e abre um novo, totalmente vazio. Você 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 salvo, além do conjunto aberto no momento. Se passar o seu mouse sobre o símbolo do coração, irá obter algo como isto:



Isto mostra-lhe que o conjunto de trabalho correspondente contém dois arquivos e os seus nomes de projetos correspondentes: `Makefile` e `alteracoes.h`. Se clicar em **Carregar**, irá fechar e salvar o conjunto de trabalho atual (que aparece aqui com os arquivos `tria.h` e `tria.cc` abertos) e irá abrir o conjunto selecionado. Você poderá também excluir de forma permanente um conjunto de trabalho, o qual o irá remover da lista de conjuntos de trabalho salvos.

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 arquivo: insira parte do nome do arquivo e selecione entre todos os arquivos das pastas dos projetos da sessão atual que correspondam ao texto; assim, será aberto o arquivo
Ctrl-Alt-C	Abrir rapidamente a classe: insira parte do nome de uma classe e selecione entre todas as classes que corresponderem; o cursor irá então saltar para a declaração da classe
Ctrl-Alt-M	Abrir rapidamente a função: insira parte do nome de uma função (membro) e selecione entre todos os nomes que corresponderem; repare que a lista mostra tanto as declarações como as definições, e o cursor irá então saltar para o item selecionado
Ctrl-Alt-Q	Abertura rápida universal: digite qualquer coisa (nome de um arquivo, 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 acontecendo neste arquivo, por exemplo declarações de classes e definições das funções

Ctrl-,	Ir para a definição de uma função, caso o cursor esteja no momento sobre a declaração de uma função
Ctrl-.	Ir para a declaração de uma função ou variável, caso o cursor esteja no 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	Localizar próxima
Ctrl-R	Substituir
Ctrl-Alt-F	Pesquisa-substituição em vários arquivos

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 selecionado ou a linha atual
Ctrl-Shift-D	Comentar o texto selecionado ou a linha atual
Alt-Shift-D	Documentar a função atual. Se o cursor estiver sobre a declaração de uma função ou classe, então, ao invocar esta combinação, irá criar um comentário no estilo do 'doxygen' devidamente preenchido com uma listagem de todos os parâmetros, valores devolvidos, etc.
Ctrl-T	Trocar o caractere atual com o anterior
Ctrl-K	Apaga a linha atual (nota: esta não é a opção do 'emacs' para 'apagar daqui até ao fim da linha')

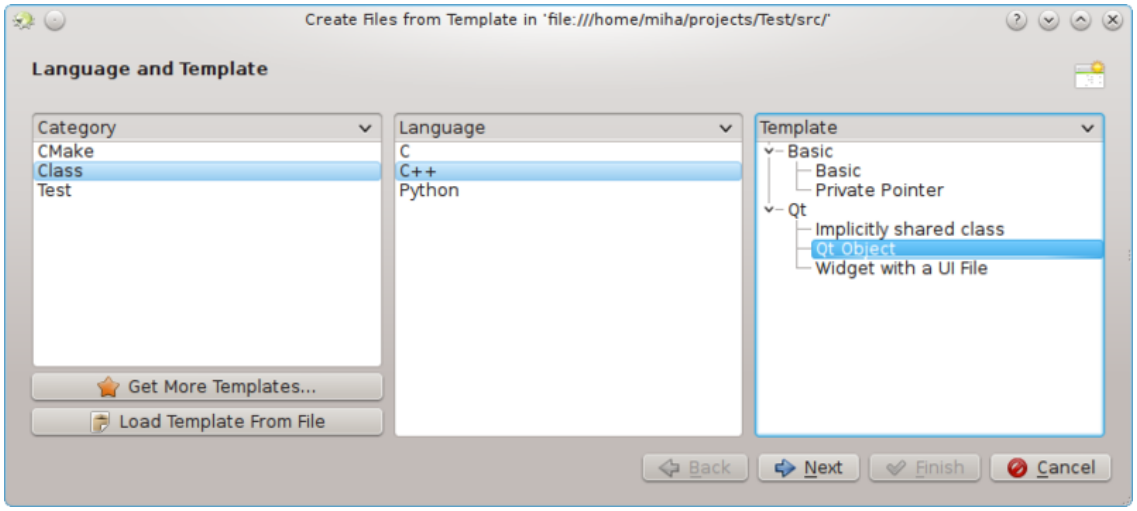
Capítulo 4

Geração de código com modelos

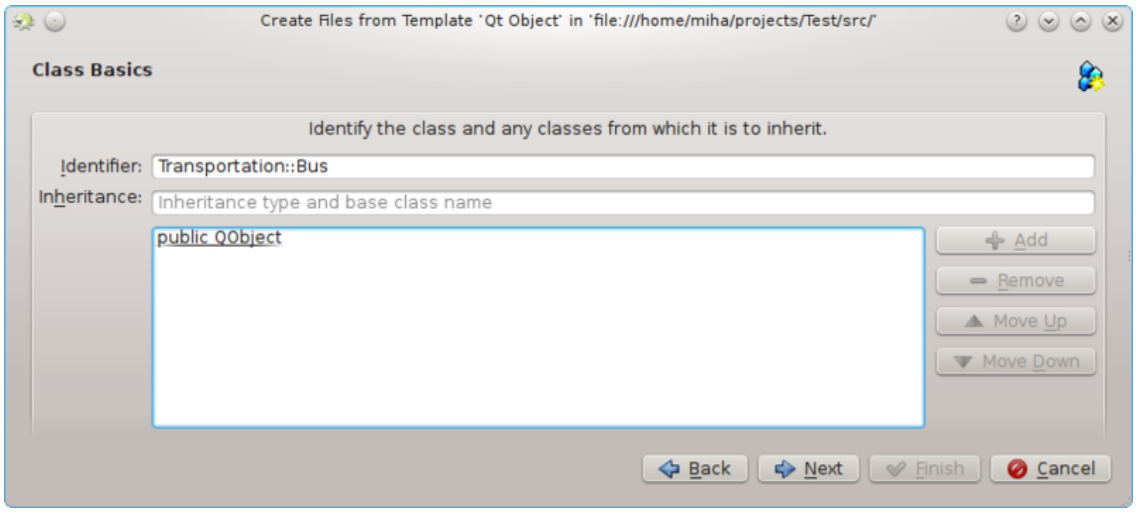
O KDevelop usa modelos para gerar arquivos de código-fonte e evitar que códigos repetitivos sejam escritos manualmente.

4.1 Criar uma nova classe

O uso mais comum para geração de código é provavelmente a escrita de novas classes. Para criar uma nova classe em um projeto existente, dê um clique direito em uma pasta do projeto e selecione **Criar a partir de um modelo...**. A mesma janela pode ser aberta a partir do menu clicando em **Arquivo** → **Novo a partir de um modelo...**, mas usar uma pasta do projeto tem a vantagem de definir uma URL base para os arquivos gerados. Selecione `Classe` na visão de seleção da categoria, e a linguagem e modelo nas outras duas visões. Após ter selecionado um modelo de classe, você terá que especificar os detalhes da nova classe.

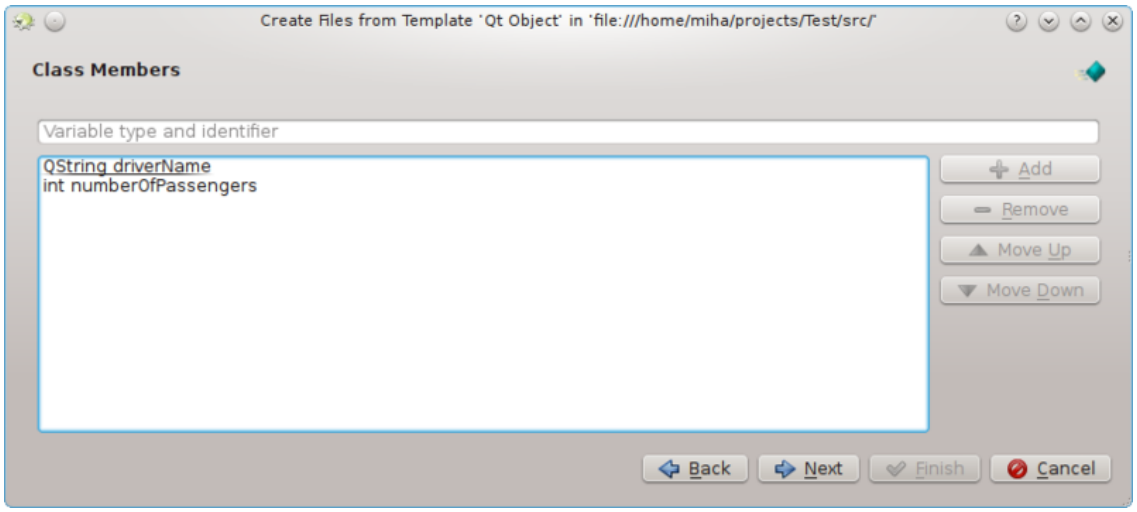


Primeiro você deve especificar um identificador para a nova classe. Isto pode ser um nome simples (como `Onibus`) ou um identificador completo com espaço de nomes (como `Transporte::Onibus`). No último caso, o KDevelop analisará o identificador e separará corretamente o espaço de nomes a partir do nome fornecido. Na mesma página, você pode adicionar uma classe base em si, bem como remover a/ou adicionar outras bases. Você deve digitar a sentença completa de herança aqui, que é dependente da linguagem, como `public QObject` para C++, `extends Algum aClasse` para PHP ou simplesmente o nome da classe para Python.



Na próxima página, você poderá selecionar métodos virtuais de todas as classes herdadas, bem como alguns construtores padrão, destrutores e operadores. Selecionando a caixa de opção ao lado da assinatura de um método, será implementado este método na nova classe.

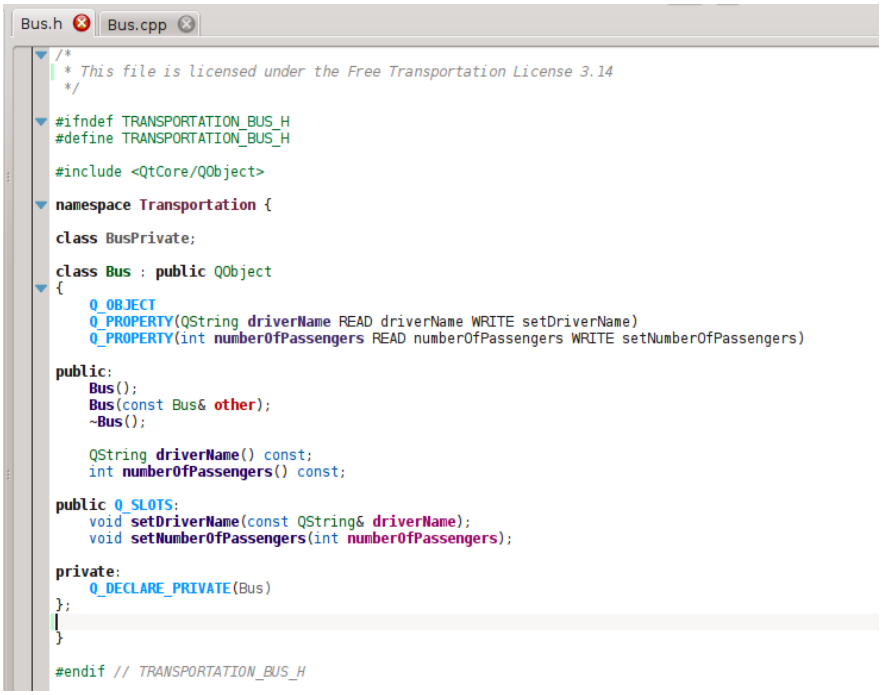
Clicar em **Próximo** leva à página aonde você pode adicionar membros à classe. Dependendo do modelo selecionado, eles podem aparecer na nova classe como variáveis de membro, ou o modelo pode criar propriedades com apontadores e obtentores para eles. Em uma linguagem aonde tipos de variável devem ser declarados, como em C++, você terá que especificar tanto o tipo como o nome do membro, como `int numero` ou `QString nome`. Em outras linguagens, você não precisa declarar o tipo, mas é uma boa prática de programação declará-lo, pois o modelo selecionado pode ainda fazer algum uso dele.



Nas páginas a seguir, você pode selecionar uma licença para sua nova classe, definir opções personalizadas necessárias do modelo selecionado, e configurar a localização de saída para todos os arquivos gerados. Ao clicar em **Concluir**, você conclui o assistente e cria a nova classe. Os arquivos gerados serão abertos no editor, assim você pode iniciar a adicionar código logo a seguir.

Após criar uma nova classe em C++, você terá a opção de adicionar a classe ao projeto alvo. Selecione um alvo a partir da página da janela, ou cancele a página e adicione os arquivos ao alvo manualmente.

Se você selecionar o modelo de `Objeto Qt`, selecione alguns dos métodos padrão, e adicione duas variáveis-membro, a saída deve se parecer com a imagem a seguir.



Você pode ver que os membros de dados são convertidos para as propriedades do Qt, com funções de acesso e macros Q_PROPERTY. Até mesmo argumentos para funções de ajuste são passados por referência à constante, quando apropriado. Adicionalmente, uma classe privada é declarada, e um ponteiro privado é criado com Q_DECLARE_PRIVATE. Tudo isto é feito pelo modelo. Selecionar um modelo diferente no primeiro passo poderá mudar completamente o resultado.

4.2 Criar um novo teste unitário

Apesar de a maioria das plataformas de teste exigirem que cada teste seja também uma classe, o KDevelop inclui um método para simplificar a criação de testes unitários. Para criar um novo teste, dê um clique-direito em uma pasta do projeto e selecione **Criar a partir de um modelo...**. Na página de seleção de modelo, selecione `Teste` como categoria, e então selecione sua linguagem de programação e modelo e clique em **Próximo**.

Será solicitado o nome do teste e uma lista de casos de testes. Para os casos de testes, você só terá que 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 determinado prefixo especial. No KDevelop, o modelo é responsável pela adição do prefixo, assim você não terá que indicá-lo aqui nos casos de teste. Depois de clique em **Próximo**, especifique a licença e os locais de saída dos arquivos gerados, para que o teste seja depois criado.

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

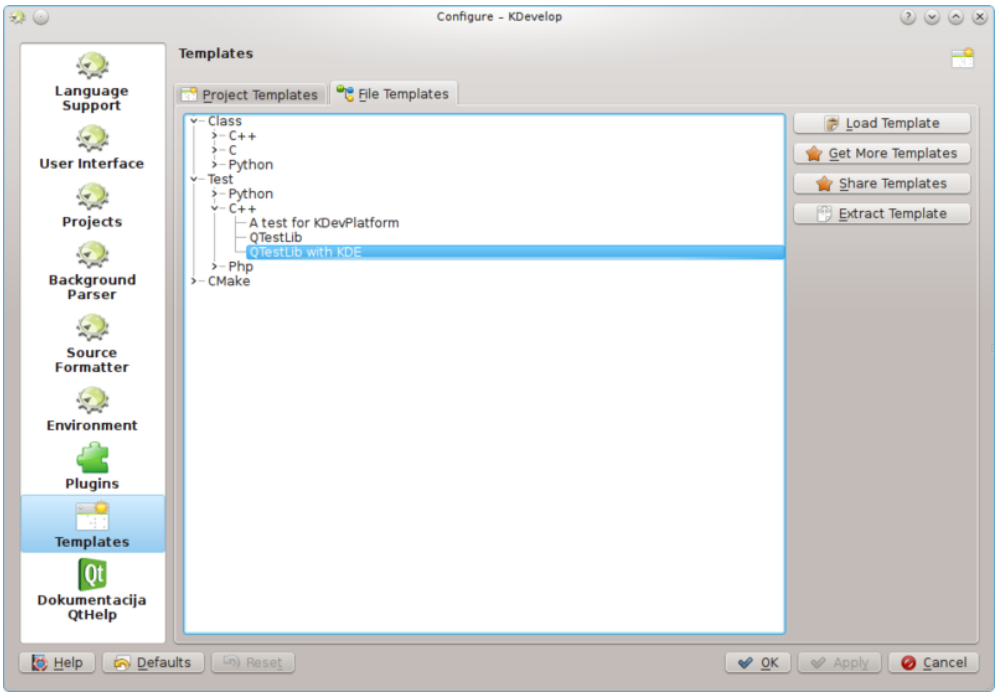
4.3 Outros arquivos

Enquanto classes e unidades de teste recebem uma atenção especial ao gerar o código a partir de modelos, o mesmo método pode ser usado para qualquer tipo de arquivo de código-fonte. Por exemplo, alguém poderia usar um modelo para um módulo de busca do CMake ou um arquivo .desktop. Isto pode ser feito selecionando **Criar a partir de um modelo...**, e selecionando a categoria desejada e modelo. Se a categoria selecionado não for `Classe` ou `Teste`, você terá somente

a opção de selecionar a licença, quaisquer opções especificadas pelo modelo, e a localização de saída dos arquivos. Da mesma maneira procedida com classes e testes, terminar o assistente irá gerar os arquivos e abri-los no editor.

4.4 Gerenciando modelos

A partir do assistente **Arquivo** → **Novo a partir de modelo...**, você pode também baixar arquivos de modelo adicionais clicando no botão **Baixar mais modelos...**. Isto abre a janela para baixar novidades, de onde você pode instalar modelos adicionais. bem como atualizá-los ou removê-los. Existe também um módulo de configuração para modelos, que pode ser acessado clicando em **Configurações** → **Configurar o KDevelop** → **Modelos**. A partir dele, você pode gerenciar tanto modelos de arquivo (explicado acimo) como modelos de projeto (usado para criar projetos novos).



É claro, se nenhum dos modelos disponíveis for adequado para o seu projeto, você sempre pode criar novos. A maneira mais fácil é provavelmente copiar e modificar um modelo existente, enquanto este rápido tutorial e o documento de especificação mais extenso estão aqui para auxiliá-lo. Para copiar um modelo instalado, abra o gerenciador de modelos clicando em **Configurações** → **Configurar o KDevelop...** → **Modelos**, selecione o modelo que deseja copiar, e então clique no botão **Extrair modelo**. Selecione uma pasta de destino, e então clique em **OK**, e o conteúdo do modelo será extraído para a pasta selecionada. Agora você pode editar o modelo abrindo os arquivos extraídos e modificando-os. Após terminar, você pode importar seu novo modelo para o KDevelop abrindo o gerenciador de modelos, ativando a aba correspondente (seja a de **Modelos de Projeto** ou **Modelos de Arquivo**) e clicando em **Carregar modelo**. Abra o arquivo de descrição do modelo, que é o que possui o sufixo `.kdevtemplate` ou `.desktop`. O KDevelop irá comprimir os arquivos em um pacote de modelo e importar o modelo.

> NOTA
> Ao copiar um modelo existente, certifique-se de renomeá-lo antes de importá-lo novamente. Caso contrário, você sobrescreverá o modelo antigo, ou acabará com dois modelos com nomes idênticos. Para renomear um modelo, renomeie o arquivo de descrição para algo único (mas mantenha o sufixo), e mude a entrada `Nome` no arquivo de descrição.

Se você deseja escrever um modelo a partir do zero, você pode iniciar com um modelo de classe C++ de exemplo criando um novo projeto e selecionando o projeto `Modelo de classe C++` na categoria `KDevelop`.

Capítulo 5

Compilar os projetos com Makefiles personalizados

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

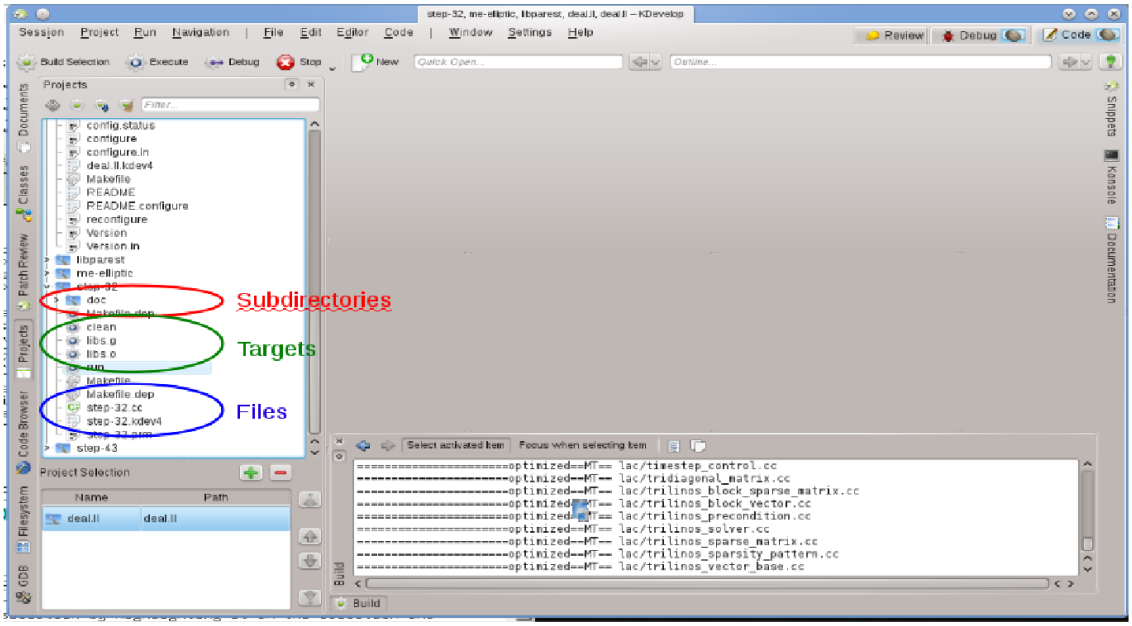
> NOTA
> O KDevelop 4.x não tem nenhum suporte às **'autotools' da GNU** no momento em que esta seção foi escrita. Se o seu projeto as usar, terá que rodar o `./configure` ou qualquer um dos outros comandos relacionados à mão, numa linha de comando. Se quiser fazer isto dentro do KDevelop, abra a ferramenta 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 comando e poderá então executar o `./configure` a partir da linha de comando nesta janela.

O primeiro passo é indicar ao KDevelop quais são os alvos nos seus arquivos Makefile. Existem duas formas de selecionar os alvos do Makefile individualmente e escolher uma lista dos que deseja compilar com mais frequência. Em ambas as abordagens, abra a ferramenta de **Projetos**, clicando no botão de **Projetos** no perímetro da janela principal do KDevelop (se não tiver este botão, veja como adicionar um botão destes aí). A janela da ferramenta de **Projetos** tem duas partes: a metade superior — chamada **Projetos** — apresenta todos os seus projetos e permite-lhe expandir as árvores de pastas subjacentes. A metade inferior — chamada **Seleção dos projetos** — apresenta um subconjunto desses projetos que serão compilados ao escolher o item do menu **Projeto** → **Compilar a seleção** ou pressionar **F8**; voltaremos a esta parte mais adiante.

5.1 Compilar os alvos individuais do Makefile

Na parte superior da área do projeto, expanda a subárvore de um projeto, por exemplo o projeto onde deseja executar um alvo em particular do Makefile. Isto fornecerá ícones para (i) as pastas sob este projeto, (ii) os arquivos na pasta de topo deste projeto, (iii) os alvos do Makefile que o KDevelop consegue identificar. Estas categorias aparecem na imagem à direita. Lembre-se de 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, clique nele com o botão direito do mouse e selecione **Compilar**. Por exemplo, se fizer isto com o alvo 'clean' (limpar), irá simplesmente executar o comando 'make clean'. Você poderá ver isto acontecendo na subjanela **Compilação** que aparece, mostrando-lhe o comando e o seu resultado. (Esta janela corresponde à ferramenta para **Compilar**, assim você 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 Selecionar uma coleção de alvos do Makefile para uma compilação repetida

Se clicar com o botão direito em alvos individuais do Makefile, sempre que quiser compilar algo, irá perder um tempo precioso. Em vez disso, será bom ter alvos individuais para um ou mais projetos da sessão que possa então compilar de forma repetida sem muito trabalho com o mouse. Aí é onde o conceito das 'Seleções de alvos de compilação' pode ajudar: é uma coleção de alvos dos arquivos Makefile que são executados um após o outro quando clicar no botão **Compilar a seleção** na lista de botões do topo, selecionar a opção do menu **Projeto** → **Compilar a seleção** ou pressionar a tecla de função **F8**.

A lista com os alvos selecionados da Makefile aparece na metade inferior da área de **Projetos**.

Por padrão, a seleção contém todos os projetos, mas você poderá alterar isso. Por exemplo, se a sua lista de projetos tiver três destes (uma biblioteca de base L e duas aplicações A e B), mas você só estiver trabalhando no momento no projeto A, você poderá querer remover o projeto B da seleção, selecionando-o nessa lista e clicando no botão . Além disso, você poderá querer garantir que a biblioteca L é compilada antes do projeto A, movendo os itens da seleção para cima ou para baixo com os botões à direita da lista. Também poderá obter um alvo da Makefile em particular para a seleção se clicar com o botão direito sobre ela e selecionar **Adicionar ao conjunto de compilação** ou simplesmente selecioná-lo e clicar em , logo acima da lista de alvos selecionados.

O KDevelop permite-lhe configurar o que desejar sempre que compilar a seleção. Para isso, use a opção do menu **Projeto** → **Abrir a configuração**. Aí, poderá por exemplo selecionar 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 seleção.

5.3 O que fazer com as mensagens de erro

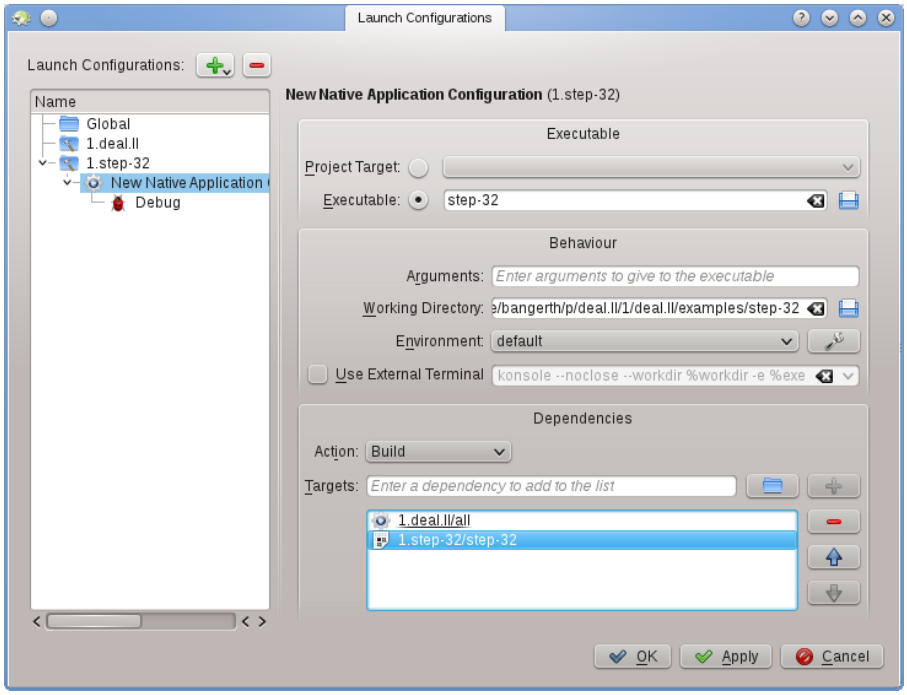
Se o compilador encontrar uma mensagem de erro, basta clicar na linha com a mensagem de erro para que o editor vá para a linha (e, se possível, a coluna) onde foi comunicado o erro. Dependendo da mensagem de erro, o KDevelop poderá oferecer-lhe várias ações possíveis para corrigir o erro, como por exemplo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

Capítulo 6

Executar os programas no KDevelop

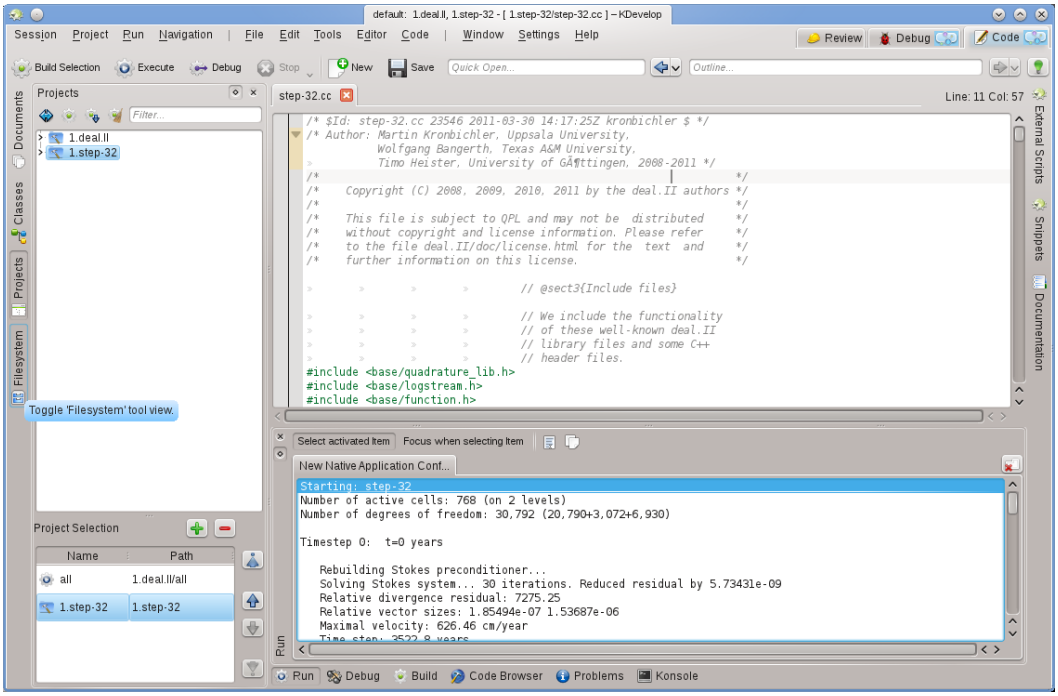
Assim que tiver compilado um programa, você irá querer executá-lo. Para isso, é necessário configurar *Lançamentos* para os seus projetos. Um *Lançamento* consiste no nome de um executável, um conjunto de parâmetros da linha de comando e um ambiente de execução (como por exemplo 'executar este programa 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**, selecione o projeto para o qual deseja adicionar um lançamento e clique no botão . Depois, insira o nome do executável e o local onde deseja executá-lo. Se a execução do programa depender da compilação do executável e/ou das suas bibliotecas em primeiro lugar, então poderá adicioná-los à lista no fundo: selecione **Compilar** no menu e depois clique no símbolo à direita do

campo de texto e selecione o alvo que deseja ter compilado. No exemplo acima, foi selecionado o alvo **all** (tudo) do projeto *1.jogada.II* e *passo-32* do projeto *1.passo-32* para se certificar que tanto a biblioteca de base como o programa foram compilados e estão atualizados antes de executar o programa em si. Já que está aqui, você poderá também configurar um lançamento de depuração, clicando para isso no símbolo **Depuração** e adicionando o nome do programa de depuração; se este for o depurador padrão do sistema (por exemplo o gdb no Linux®), então não terá que efetuar este passo.



Você poderá agora tentar executar o programa: Selecione **Executar** → **Executar o lançamento** a partir do menu da janela principal do KDevelop (ou pressionar **Shift-F9**), para que o seu programa se execute em uma subjanela separada do KDevelop. A imagem acima mostra o resultado: a nova subjanela da ferramenta **Executar**, no fundo, mostra o resultado do programa que está sendo executado, neste caso, do programa *passo-32*.

> NOTA
> Se você tiver configurado vários lançamentos, poderá escolher qual deseja executar quando pressionar **Shift-F9**, indo à opção **Executar** → **Configuração de lançamento atual**. Existe uma forma não óbvia de editar o nome de uma configuração: na janela que obtém quando selecionar a opção **Executar** → **Configuração de lançamento atual**, faça duplo-clique 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

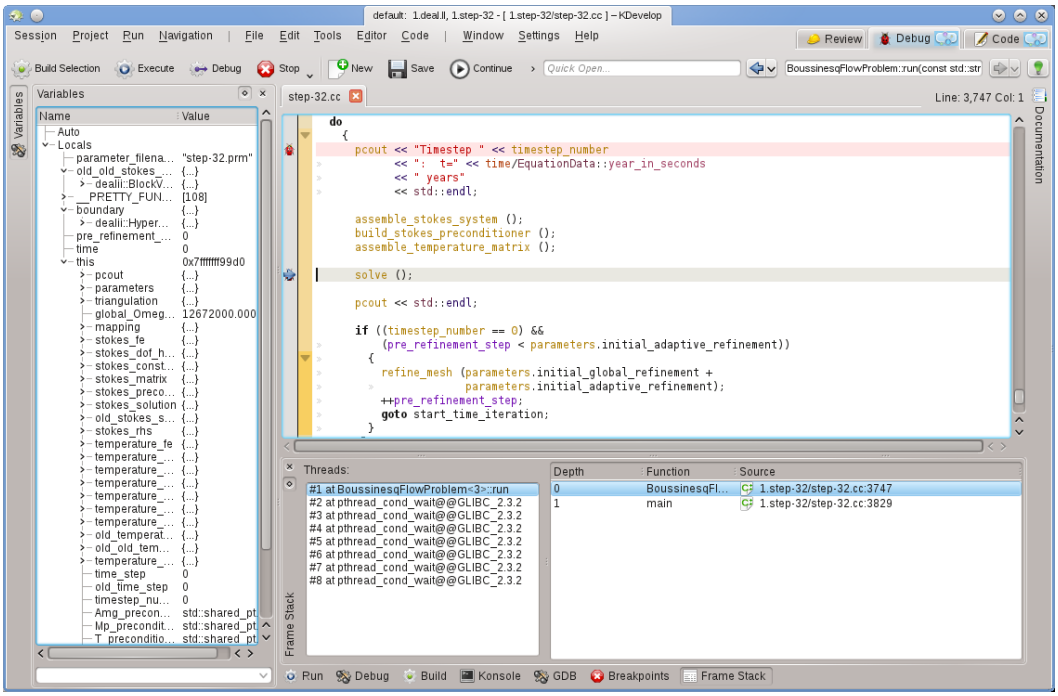
Alt-F9	Executar o programa no depurador; você poderá querer definir pontos de parada de antemão; por exemplo, se clicar com o botão direito do mouse numa linha em particular do código-fonte

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 poderá executá-lo num depurador: Selecione o item do menu **Executar** → **Depurar o lançamento** ou pressione **Alt-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 `Executar`. Isto significa que, caso o programa invoque o `abort()` em algum ponto (por exemplo quando você chegar a uma assertiva mal-sucedida) ou se tiver um erro de segmentação (proteção 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 parada sobre todas essas linhas da sua base de código onde deseja que o depurador pare, antes de executar o lançamento de depuração. Você poderá fazer isso com a opção do menu **Executar** → **Comutar o ponto de parada** ou se clicar com o botão direito sobre uma linha e selecionar a opção **Comutar o ponto de parada** do menu de contexto.

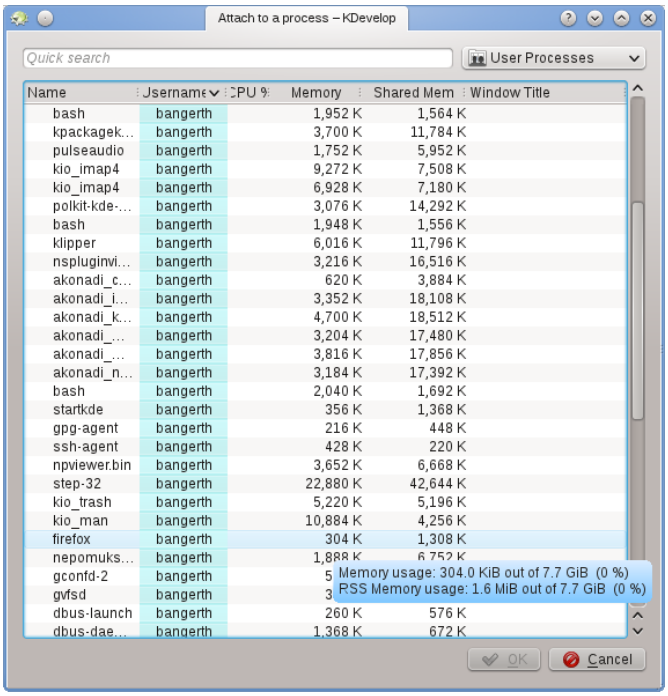


A execução de um programa no depurador irá colocar o KDevelop num modo diferente: irá substituir todos os botões de 'Ferramentas' no perímetro da janela principal por outros que sejam apropriados para a edição. Você poderá ver qual dos modos em que se encontra se olhar para o canto superior direito da janela: existem páginas chamadas **Revisão**, **Depuração** e **Código**; se clicar 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 seção Ferramentas e janelas.

Assim que o depurador parar (num ponto de parada ou num ponto em que a função `abort()` seja chamada), você poderá inspecionar uma grande quantidade de informações sobre o seu programa. Por exemplo, na imagem acima, foi selecionada 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 a execução chegou ao ponto de parada atual à direita (aqui: o `main()` invocou o `executar()`; a lista seria maior se tivesse parado numa função chamada pelo próprio `executar()`). À esquerda, você poderá inspecionar as variáveis locais, incluindo o objeto atual (o objeto referenciado pela variável `this`).

A partir daqui, existem várias possibilidades disponíveis: você poderá executar a linha atual (**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 atualiza as variáveis apresentadas à esquerda para os seus valores atuais. Você poderá também passar o mouse sobre um símbolo no seu código, por exemplo uma variável; o KDevelop irá então mostrar o valor atual 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 clicar no botão da ferramenta **GDB** no fundo e ter a possibilidade de introduzir diretamente comandos do gdb, por exemplo para alterar o valor de uma variável (possibilidade para a qual não existe no 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 isso, vá para a opção do menu **Executar** → **Anexar ao processo**, a qual irá abrir uma janela como a anterior. Você irá querer selecionar o programa que corresponde ao seu projeto aberto no 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. Você poderá simplificar a sua vida se for à lista no canto superior direito da janela. O valor padrão é **Processos do usuário**, isto é todos os programas que são executados por qualquer um dos usuários autenticados no momento nesta máquina (se este for o seu computador pessoal ou portátil, provavelmente você será o único usuário de fato, além do 'root' e das várias contas de serviços); a lista não inclui os processos executados pelo usuário 'root', contudo. Você poderá limitar a lista se escolher a opção **Processos próprios**, removendo todos os programas executados pelos outros usuários. Melhor ainda, selecione a opção **Apenas os programas**, a qual retira muitos dos processos que estão sendo executados com o seu nome, mas com os quais não interage normalmente, como o gerenciador de janelas, as tarefas de segundo plano e assim por diante, as quais não são normalmente candidatas para a depuração.

Assim que tiver selecionado 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 parada, 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')

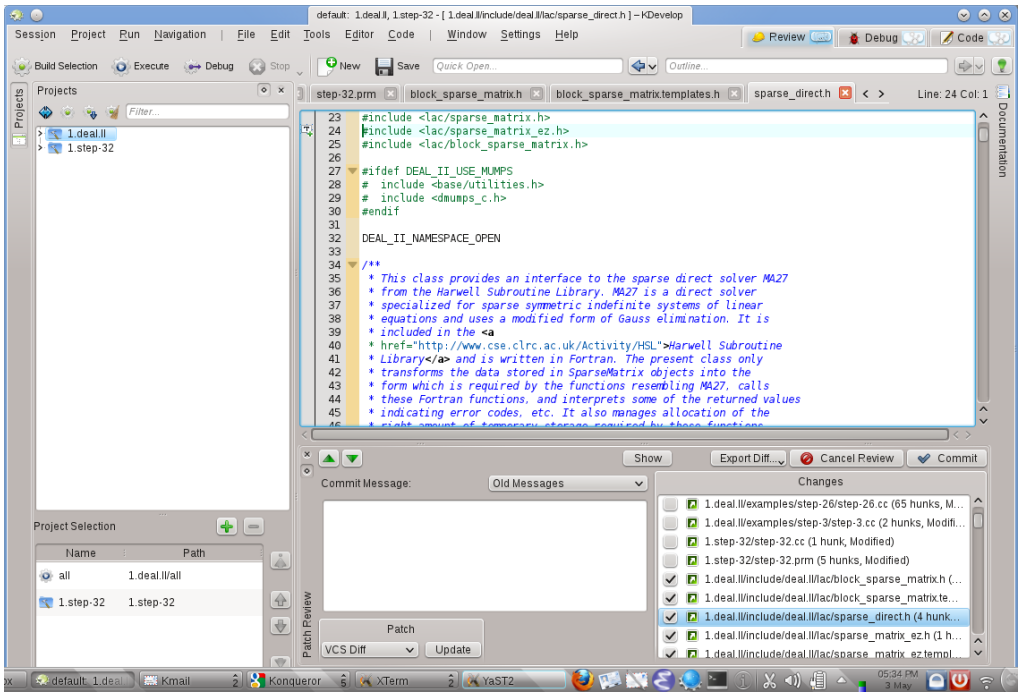
Capítulo 8

Lidar com sistemas de controle de versões

Se você estiver lidando com projetos maiores, será provável que o código-fonte seja gerenciado por um sistema de controle 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 controle de versões suportado qualquer.

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

- Se a sua pasta estiver desatualizada, você poderá atualizá-la a partir do repositório: Clique no nome do projeto com o botão direito do mouse, vá ao menu **Subversion** e selecione **Atualizar**. Isto irá obter atualizações de todos os arquivos que pertençam a este projeto e que digam respeito ao repositório.
- Se quiser restringir esta ação apenas às subpastas ou arquivos individuais, então expanda a árvore deste projeto para o nível que desejar e clique com o botão direito do mouse sobre uma subpasta ou arquivo, fazendo o mesmo que se descreveu acima.



- Se você tiver editado um ou mais arquivos, expanda a área do projeto até à pasta onde se encontram estes arquivos e clique 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 arquivos nesta pasta.

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

- Se quiser enviar um ou mais arquivos editados para o servidor, clique com o botão direito sobre um arquivo individual, subpasta ou sobre o projeto todo e selecione a opção **Subversion** → **Enviar**. Isto fará mudar o modo para **Revisão**, o terceiro modo que existe 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 subpasta/projeto inteiro e cada um dos arquivos individuais alterados com as alterações realçadas (veja as várias páginas nesta parte da janela). Por padrão, todos os arquivos alterados estão no conjunto de alterações que estará prestes a enviar, mas você poderá desligar alguns dos arquivos, caso as suas modificações não estejam relacionadas com o que deseja enviar. No exemplo à direita, foi desligado o arquivo `passo-32.cc` e `passo-32.prm` porque as alterações destes arquivos não têm nada a ver com as outras que foram feitas no projeto e não se pretende enviá-las agora (poder-se-á pensar nisso num envio em separado). Depois de rever as alterações, poderá inserir uma mensagem de envio no campo de texto e clicar em **Enviar** à direita para enviar tudo o que desejar.

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

Capítulo 9

Personalizar o KDevelop

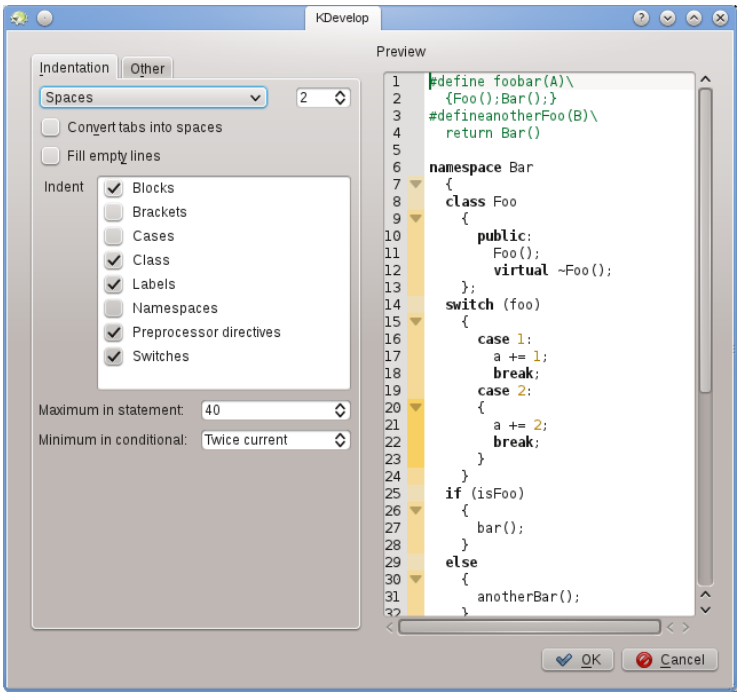
Existem momentos em que poderá querer alterar a aparência ou comportamento predefinidos do KDevelop, por exemplo, se estiver habituado a combinações de teclas diferentes ou porque o seu projeto necessita de um estilo de indentação diferente para o código-fonte. Nas seções a seguir, iremos discutir de forma breve as diferentes formas com que pode personalizar o KDevelop para esses fins.

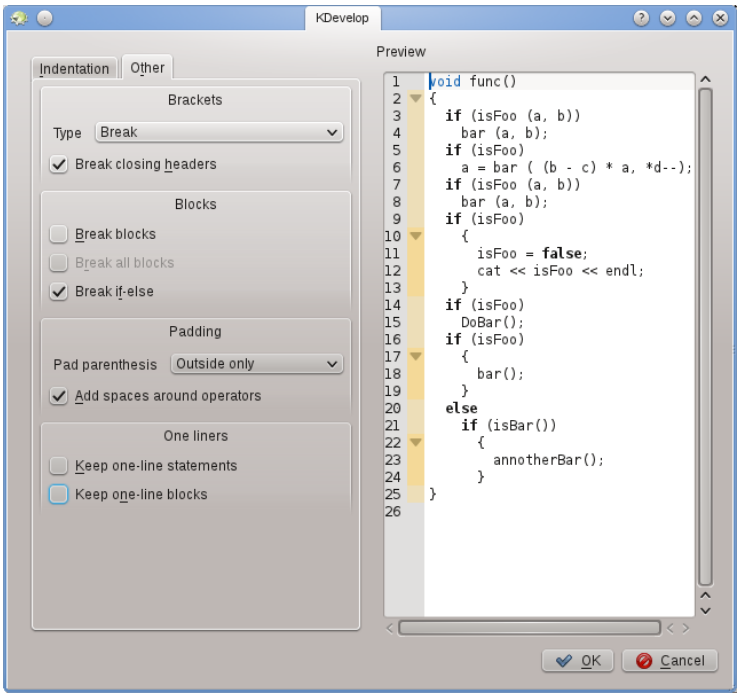
9.1 Personalizar o editor

Existe um conjunto de coisas úteis que poderá configurar no editor incorporado do KDevelop. A mais universal será ativar 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 ativar 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 parada na linha atual.

9.2 Personalizar a indentação do código

Muitos de nós gostamos do código formatado de uma determinada forma. Muitos projetos também obrigam a um determinado 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 **Configurações** → **Configurar o KDevelop**, depois clique no **Formatador de código** à esquerda. Você 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. Você poderá não haver uma forma de recriar exatamente o estilo com que o código do seu projeto 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 abaixo.


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

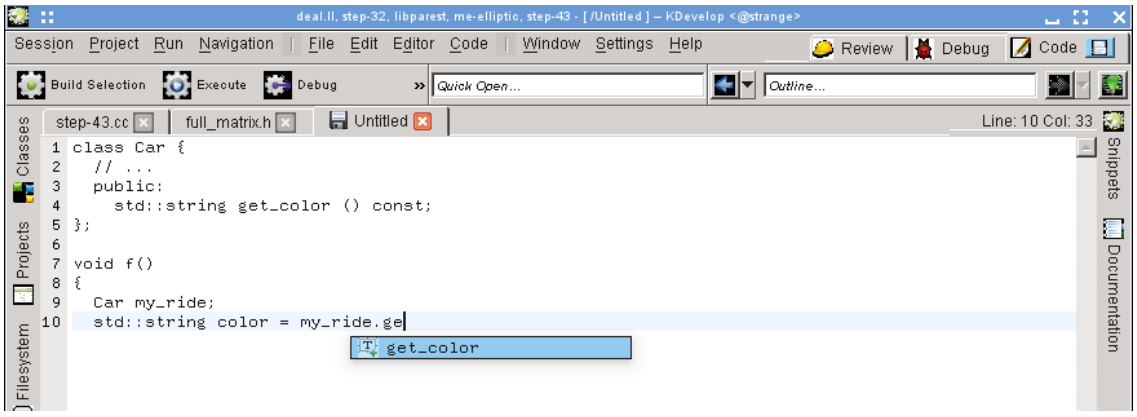
9.3 Personalizar os atalhos de teclado

O KDevelop tem uma lista quase ilimitada de combinações de teclas (algumas delas encontram-se nas 'seções de combinações úteis de teclas' de vários capítulos neste manual) que poderão ser alteradas a seu gosto no menu **Configurações** → **Configurar atalhos**. No topo da janela, você 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 inserem tabulações à mão e assim preferem que o editor escolha o layout do mesmo; com o atalho alterado, pressionar **Tab**, fará com que o KDevelop indente/retire a indentação/alinhe o código). A segunda é associar o **Comutar o ponto de parada** ao **Ctrl-B**, uma vez que é uma operação bastante frequente.

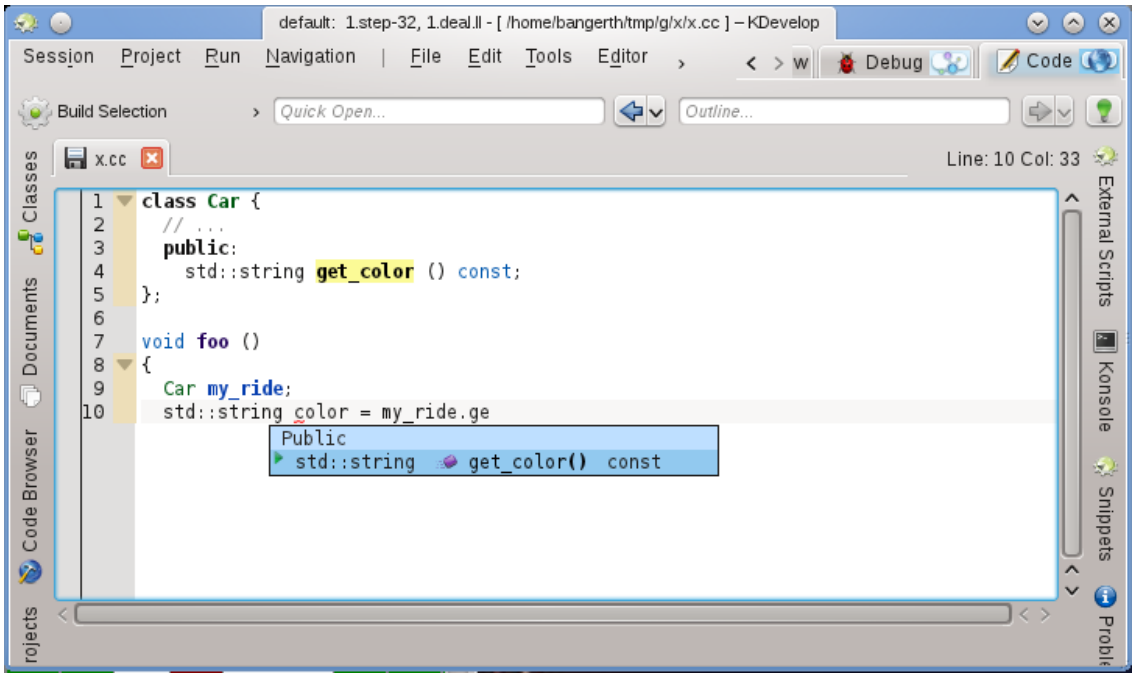
9.4 Personalizar a complementação automática do código

A complementação do código é discutida na seção 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 ambiente do KDE e fornece a complementação automática com base nas palavras que já tiverem sido vista em outras partes do mesmo documento. Essa complementação automática poderá ser identificada na dica pelo ícone que a antecede:



A complementação de código do editor poderá ser personalizada com a opção **Configurações** → **Configurar o editor** → **Edição** → **Complementação automática**. Em particular, você pode selecionar quantos caracteres necessita digitar para que a janela de complementação automática apareça.

Por outro lado, a complementação automática própria do KDevelop é muito mais poderosa, uma vez que tem em conta a informação semântica sobre o contexto. Por exemplo, ele sabe que funções-membro deverão oferecer quando escrever `objeto.`, etc., como demonstrado acima:



Esta informação de contexto vem de vários plugins de suporte às linguagens, os quais poderão ser utilizados depois de um determinado arquivo ter sido salvo (para que possa então verificar o tipo de arquivo e usar o suporte da linguagem correto).

A complementação do KDevelop está configurada para aparecer assim que digitar, praticamente em todo local onde seja possível completar algo. Isto é configurável na opção **Configurações** → **Configurar o KDevelop** → **Suporte à linguagem**. Se não estiver já definido (como deveria, por padrão), certifique-se de que a opção **Ativar a invocação automática** está ativa.

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

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

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

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

- **Sempre a complementação mínima**: Nunca mostrar a 'Complementação total'
- **Complementação automática mínima**: Só mostrar a 'Complementação total' quando esta tiver sido invocada manualmente (isto é, quando pressionar **Ctrl-Espaço**)
- **Sempre a complementação total**: Mostrar sempre a 'Complementação total'

Capítulo 10

Créditos e licença

Tradução de Marcus Gama marcus.gama@gmail.com e André Marcelo Alvarenga alvarenga@kde.org