Handboek van KDevelop

Deze documentatie is geconverteerd vanuit de KDE UserBase KDevelop4/Manual pagina.
Vertaler/Nalezer: Freek de Kruijf
Vertaler: Ronald Stroethoff

Inhoudsopgave

Samenvatting

KDevelop is een geïntegreerde ontwikkelomgeving om te worden gebruikt voor een brede variëteit van programmeertaken.

Hoofdstuk 1

Wat is KDevelop?

KDevelop is een moderne geïntegreerde ontwikkelomgeving (IDE) voor C++ (en andere programmeertalen) en is een van de vele KDE programma's. Het draait op Linux® (zelfs als u een andere desktop, zoals GNOME, gebruikt) maar is ook beschikbaar voor de meeste andere varianten van UNIX® en zelfs voor Windows.

KDevelop heeft alle voorzieningen van een moderne IDE. Voor grote projecten en programma's is het belangrijk dat KDevelop *C++ begrijpt*: het verwerkt de gehele broncode en onthoudt welke member-functies van welke klassen zijn, waar variabelen gedefinieerd zijn, welke types het zijn en vele andere dingen over uw code. Als bijvoorbeeld: in een van de headers in uw project wordt een klasse gedeclareerd

```
class Car {
  // ...
  public:
    std::string get_color () const;
};
```

en later in uw programma hebt u

```
Car my_ride;
// ...doe iets met deze variabele...
std::string color = my_ride.ge
```

heeft het bijgehouden dat `my_ride` in de laatste regel een variabele van het type `Car` is en biedt aan om automatisch `ge` aan te vullen tot `get_color()` omdat deze de enige member-functie van de klasse `Car` is die op deze manier start. In plaats doorgaan met het volledig invoeren drukt u gewoon op **Enter** om het complete woord te krijgen; dit bespaart typen, vermijdt typefouten, en u hoeft ook niet meer de exacte namen van honderden functies en klassen uit een project te onthouden.

Als een tweede voorbeeld, neemt u aan dat u een code als deze hebt:

```
double foo ()
{
  double var = my_func();
 return var * var;
}
double bar ()
{
  double var = my_func();
  return var * var * var;
}
```

Als u met uw muis boven het symbool `var` in de functie `bar` zweeft dan heeft u de mogelijkheid om alle locaties te zien waar dit symbool gebruikt is. Als u erop klikt, ziet u alleen het gebruik van deze variabele in de functie `bar` omdat KDevelop begrijpt dat de variabele `var` in de functie `foo` daarmee niets van doen heeft. Vergelijkbaar, kunt na het klikken met de rechtermuisknop de naam van de variabele wijzigen; maar dit gebeurt dan alleen met de variabele in `bar` maar niet die met dezelfde naam in `foo`.

Maar KDevelop is niet alleen een intelligente bewerker van broncode; KDevelop kan ook andere taken goed uitvoeren. het markeert de broncode in verschillende kleuren; De inspringingen die het maakt zijn instelbaar; het heeft een geïntegreerde interface voor de GNU debugger gdb; het kan u de documentatie van een functie tonen als u met de muis boven een locatie zweeft waar deze functie in gebruik is; het kan overweg met verschillende soorten build environments en compilers (bijv. met op **make** en **cmake**-gebaseerde projecten), en vele andere handige dingen die in deze handleiding beschreven worden.

Hoofdstuk 2

Sessies en projecten: De basis elementen van KDevelop

In deze sectie behandelen we sommige begrippen van hoe KDevelop de wereld ziet en hoe het werk structureert. We zullen met name de begrippen *sessies* en *projecten* uitleggen en hoe u de projecten opzet die u wilt bewerken in KDevelop.

2.1 Terminologie

KDevelop heeft de begrippen *sessies* en *projecten*. Een sessie bevat alle projecten die iets met elkaar gemeen hebben. Voor de voorbeelden die hierna komen nemen we aan dat u de ontwikkelaar bent van zowel een bibliotheek als het programma dat daarvan gebruik maakt. U kunt daarbij voor de eerste denken aan de core KDE bibliotheken en voor het laatste aan KDevelop. Ander voorbeeld: laten we aannemen dat u een Linux® kernel hacker bent en u werkt ook aan een device driver voor Linux® die nog niet is opgenomen in de kernel-tree.

Laten we de laatste als voorbeeld nemen, u heeft een sessie in KDevelop met daarin twee projecten: de Linux® kernel en de device driver. U zal deze in een enkele sessie willen hebben (en niet twee sessies met elk een enkel project) omdat het handig is om de kernel functies en data structures in KDevelop te zien wanneer u broncode voor de driver aan het schrijven bent — bijvoorbeeld omdat dan kernel functies en variabele-namen automatisch aangevuld krijgt, maar ook omdat u dan de documentatie van de kernelfunctie te zien krijgt tijdens het hacken aan de device driver.

Stelt u zich voor dat u ook een KDE ontwikkelaar bent. Dan is het verstandig om een tweede sessie te hebben die KDE als project heeft. U zou in principe maar één sessie hoeven te hebben voor dit alles , maar daar is niet echt een reden voor: in uw KDE werk, hoeft u geen toegang te hebben tot kernel of device driver functies; en u wilt geen KDE class-namen automatisch aangevuld hebben tijdens het werk aan de Linux®-kernel. Tenslotte, het compileren van een van de KDE libraries is onafhankelijk van het hercompileren van de Linux® kernel (maar het is wel verstandig om de Linux® kernel te hercompileren als u de device driver compileert omdat een of meer van de header-bestanden gewijzigd kunnen zijn).

Tenslotte, een ander gebruik van sessies is als u zowel aan de huidige ontwikkelversie van een project als aan een branch werkt: u wilt in dat geval niet dat KDevelop in de war komt met de klassen die bij de hoofdtak horen en die bij de branch horen, Het is daarom verstandig om twee sessies te hebben, met dezelfde verzameling projecten maar uit andere mappen (overeenkomend met verschillende ontwikkel-branches).

2.2 Opzetten van een sessie en importeren van een bestaand project

Laten we het bij het voorbeeld van de Linux® kernel en de device driver houden — u wilt misschien deze twee voorbeelden vervangen door uw eigen set bibliotheken of projecten. Om een nieuwe sessie voor deze twee projecten aan te maken, gaat u in het menu naar **Sessie** → **Nieuwe sessie starten** linksboven (of, als dit de eerste keer is dat u KDevelop gebruikt: gebruik gewoon de standaard sessie die u krijgt bij het eerste gebruik, deze is leeg).

We willen vervolgens deze sessie vullen met projecten waarvan we even aannemen dat ze al ergens aanwezig zijn (het aanmaken van nieuwe lege projecten wordt elders in deze handleiding beschreven). Hiervoor zijn er eigenlijk twee methoden, de keuze is afhankelijk van of het project al ergens op uw harde schijf aanwezig is of dat nog moet worden gedownload van een server.

2.2.1 Optie 1: Een project importeren Uit een versiecontrolsysteemserver

We gaan ervan uit dat het project dat we willen gaan opzetten — de Linux®-kernel — nog in een versiebeheersysteem op een server aanwezig is en dat u het nog niet heeft uitgecheckt naar uw lokale harde schijf. Ga in dat geval naar het menu **Project** om de Linux® kernel als project in de huidige sessie aan te maken, volg daarna de volgende stappen:

- Ga naar **Project** → **Project ophalen** om een project te importeren
- Er zijn meerdere manieren om in de huidige sessie een nieuw project te beginnen , afhankelijk van waar de bronbestanden vandaan komen: u kunt eenvoudig in KDevelop een bestaande map aanwijzen (zie optie 2 hieronder), maar u kunt ook aan KDevelop vragen om de bronbestanden vanuit een repository te downloaden.
- Aannemend dat u nog geen versie hebt opgehaald (checked out):
  - In het dialoogvenster **Bron selecteren** maak een keuze uit **Uit bestandssysteem**, **Subversion**, **Git**, **GitHub** of **KDE**
  - Kies een werkmap als doel waar de broncode naar toe uitgecheckt moet worden.
  - Geef een URL op voor de locatie van de repository waar u broncode kan verkrijgen.
  - Klik op **Gaan**. Het kan zijn dat u daarna afhankelijk van de snelheid van uw verbinding en de grootte van het project vrij lang moet wachten. Helaas geeft de voortgangsbalk van KDevelop 4.2.x niet de werkelijke voortgang weer, maar u kunt wel door regelmatig via het commando

    ```
    du -sk /path/to/KDevelop/project
    ```

    op de commandoregel zien hoeveel al is gedownloaded.

OPMERKING
Het probleem van de voortgangsbalk is gemeld als KDevelop bug 256832.

OPMERKING
Tijdens dit proces krijg ik ook de foutmelding *You need to specify a valid location for the project*, welke u gerust kan negeren.

- Het vraagt u in deze map een KDevelop project-bestand te selecteren. Maar omdat u waarschijnlijk er nog geen heeft, kunt u gewoon op **Verder** klikken.
- Klik opnieuw op **Verder**
- KDevelop zal u vervolgens vragen om een project manager te kiezen. Als in dit project standaard UNIX® make bestanden aanwezig zijn dan kiest u de makefile projectmanager.
- KDevelop zal starten met opnemen van het gehele project. Dit kan opnieuw nogal wat tijd in beslag nemen om alle bestanden te doorlopen en indexen van klassen etc. te maken. Onderaan het hoofdvenster is een voortgangsbalk die de voortgang aangeeft. (Als u meerdere processor cores heeft dan kan u dit proces versnellen door naar **Instellingen** → **KDevelop instellen** te gaan en vervolgens naar **Achtergrond ontleder** links te gaan en het aantal threads voor achtergrond ontleden te verhogen.)

2.2.2 Optie 2: een project importeren die al aanwezig is op uw harde schijf

Als alternatief, als het project waaraan u wilt gaan werken al aanwezig is op uw harde schijf (Omdat bijvoorbeeld u het als tar-bestand heeft gedownload van een FTP server, omdat u het al heeft uitgecheckt van een versiebeheersysteem, of omdat het uw eigen project is dat *alleen* op uw eigen harde schijf bestaat), gebruik dan **Project** → **Projecten openen / importeren** en kies in het dialoogvenster de map waarin uw project aanwezig is.

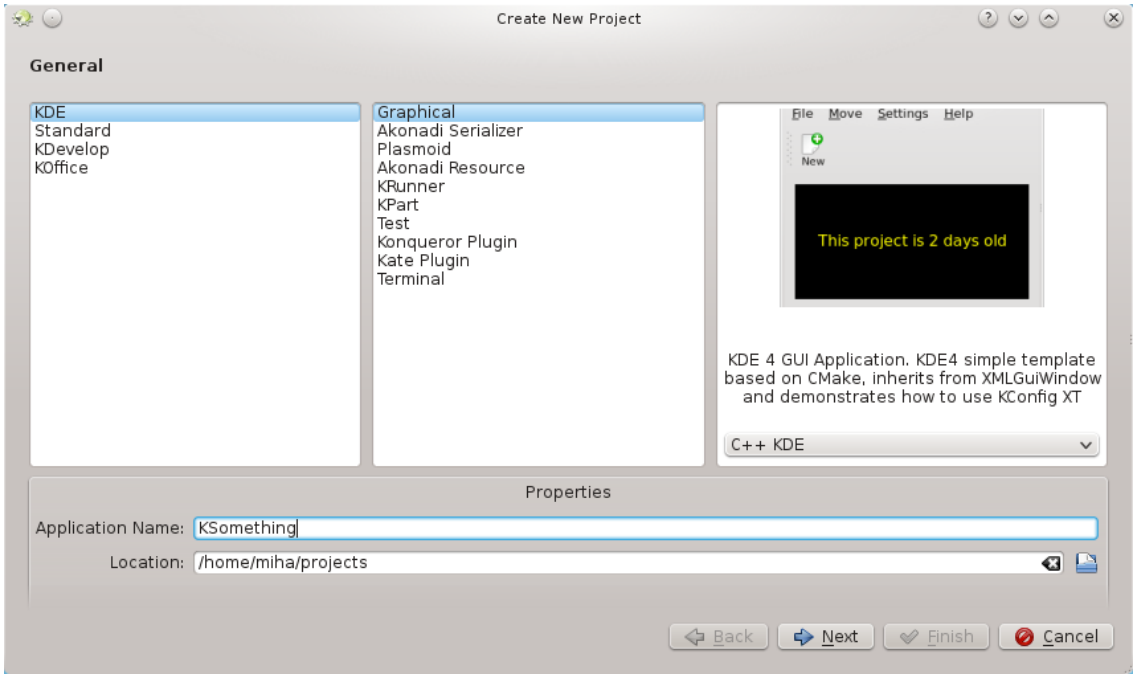
2.3 Opzetten van programma als een tweede project

Het volgende wat u wilt doen is het opzetten van andere projecten in dezelfde sessie. In het voorbeeld van hierboven wilt u de device driver als tweede project toevoegen, dit kunt u doen door exact dezelfde stappen uit te voeren.

Als u meerdere programma's of bibliotheken heeft kunt u eenvoudig de stappen herhalen om meer en meer projecten aan uw sessie toe te voegen.

2.4 Projecten vanaf nul aanmaken

Er is natuurlijk ook de mogelijkheid dat u een nieuw project vanaf nul wilt aanmaken. dit kunt u doen door gebruik van het **Project** → **Nieuw van sjabloon...** menu item, waar u een compleet keuzemenu vindt. Sommige project sjablonen zijn meegeleverd met KDevelop, maar er zijn meer beschikbaar door toepassing KAppTemplate te installeren. Kies in het dialoog het project type en de programmeertaal, geef een naam en locatie voor uw project op en klik op **Verder**.



Op de tweede pagina kunt u een versiebeheersysteem instellen. Kies het systeem dat u wilt gebruiken en vul de benodigde gegevens in. Als u geen versiebeheersysteem wilt gebruiken of u wilt het later handmatig instellen, kies dan **Geen**. Als u tevreden bent over de keuzes dan kunt u op **Voltooien** klikken.

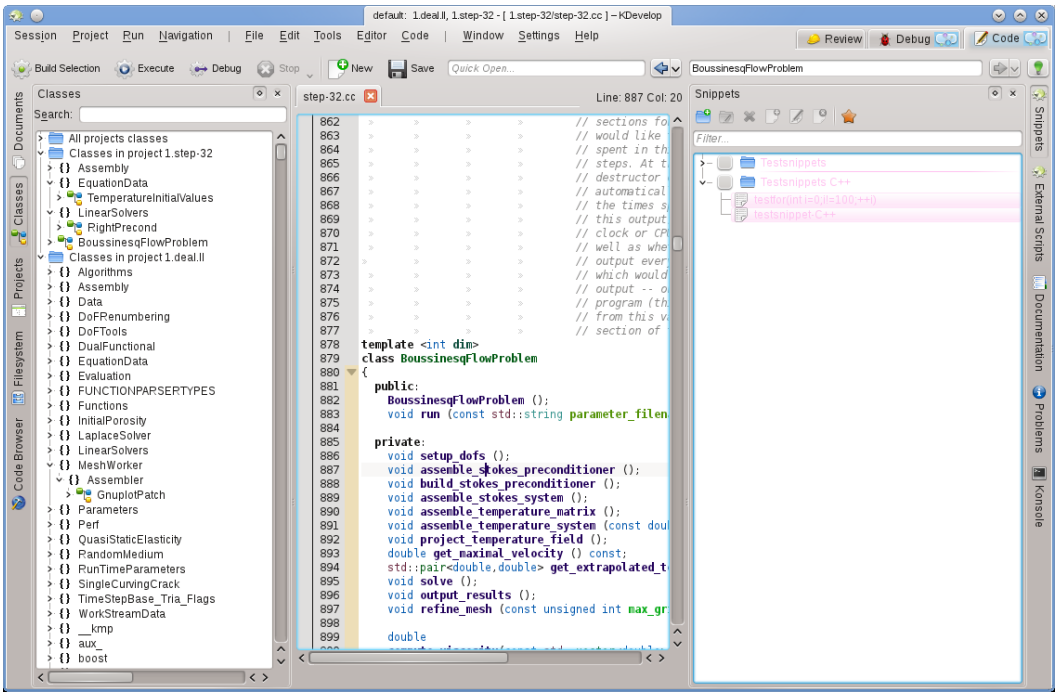
Uw project is nu klaar, u kunt daarom proberen uw project te compileren of te installeren. Sommige sjablonen zijn met commentaar in de code of zelfs een apart README bestand, en het is verstandig dat u deze eerst leest. Daarna kunt u beginnen met uw project te bewerken door welke gewenste functionaliteit dan ook toe te voegen.

Hoofdstuk 3

Broncode bewerken

Naast fout opsporing zal u de meeste tijd spenderen aan het lezen door de broncode en het schrijven daaraan tijdens het ontwikkelen van software. Om u hierin bij te staan, heeft KDevelop veel verschillende manieren om de broncode te doorzoeken en het schrijven van ervan productiever te maken. Zoals in de volgende hoofdstukken meer in detail zal worden beschreven, KDevelop is niet alleen een source editor — maar het is een source beheerssysteem waarmee u op verschillende manieren kunt kijken naar de gefilterde informatie over de bestanden die samen de broncode in uw sessie vormen.

3.1 Gereedschappen en vensters



Om met projecten te werken heeft KDevelop het concept van *hulpmiddelen*. Een hulpmiddel biedt een specifiek zicht op de hulpbron of een actie die er mee kan worden gedaan. Hulpmiddelen worden gerepresenteerd door knoppen aan de randen van uw venster (in verticale tekst langs de

linker- en rechterkant of horizontaal langs de onderkant). Als u op ze klikt, expanderen ze tot een subvenster — een *beeld* — binnen het hoofdvenster; als u nog eens klikt op de knop van het hulpmiddel, zal het subvenster weer verdwijnen.

Om een subvenster te laten verdwijnen kunt u ook klikken op de **x** rechtsboven op het subvenster

De afbeelding hierboven toont een selectie hulpmiddelen, aan de linkerkant en de rechterkant van het venster; in de afbeelding is links het **Klassen**-venster en rechts het **Fragmenten**-venster geopend met in het midden een editor met een bronbestand geopend. In de praktijk heeft u waarschijnlijk meestal alleen de editor en misschien links het **Klassen** of **Broncode Browser** venster open. Andere vensters zult u waarschijnlijk alleen tijdelijk openen als u het betreffende gereedschap gebruikt zodat er de meeste tijd meer ruimte overblijft voor de editor.

De knop **Project** zou al aanwezig moeten zijn als u KDevelop voor de eerste keer gebruikt. Klik erop: een subvenster opent waarin onderaan de projecten te zien zijn die u aan de sessie toegevoegd heeft en een lijst met mappen in de projecten bovenaan.

Er zijn vele andere hulpmiddelen die u in KDevelop kunt gebruiken, aanvankelijk zijn ze allemaal als knop aanwezig langs de rand van het hoofdvenster. Voor het toevoegen van meer knoppen gaat u naar het menu item**Venster** → **weergave van hulpmiddelen**. Hieronder zijn er enkele die u wellicht handig vindt:

- **Klassen**: een complete lijst met alle klassen die in een van de projecten uit de sessie zijn gedefinieerd met alle daarbij horende functies en variabelen. Door op een ervan te klikken opent een venster met informatie daarover.
- **Documenten**: Een lijst met de laatst gebruikte bestanden, op soort onderverdeeld (bijv. bronbestanden, patch bestanden, platte tekst documenten).
- **Broncode Browser**: Afhankelijk van de positie van uw cursor in het bestand toont dit hulpmiddel gerelateerde zaken. Bijvoorbeeld, als u op een `#include` regel staat, dan toont het informatie over het bestand die u wilt includen zoals de klassen die in dat bestand worden gedeclareerd; als u op een lege regel staat, dan toont het de in dit bestand gedeclareerde klassen en functies (alle als links: door erop te klikken gaat u naar de eigenlijke locatie in het bestand van de declaratie of definitie); als u in een functiedefinitie bent dan toont het de locatie van de declaratie met een lijst waar het gebruikt is.
- **Bestandssysteem**: Toont in een boomstructuur het bestandssysteem.
- **Documentatie**: Hier kunt u zoeken naar man pages en andere helppagina's.
- **Fragmenten**: Hier zijn stukken tekst beschikbaar die u keer op keer gebruikt en niet iedere keer opnieuw wilt invoeren. Bijvoorbeeld, in het project waarmee de afbeelding hierboven is gecreëerd, is het vaak nodig om de volgende code te creëren:

```
for (typename Triangulation< dim>::active_cell_iterator cell
      = triangulation.begin_active();
   cell != triangulation.end();
   ++cell)
```

Dit is een verschrikkelijk stuk tekst dat bijna iedere keer dat u een dergelijke loop nodig heeft er precies zo uitziet — daarom is het een goede kandidaat voor een fragment.

- **Konsole**: Opent in het hoofdvenster een venster met de commandoregel voor de zeldzame keer dat u een commando (bijv. to run `./configure`) wilt starten.

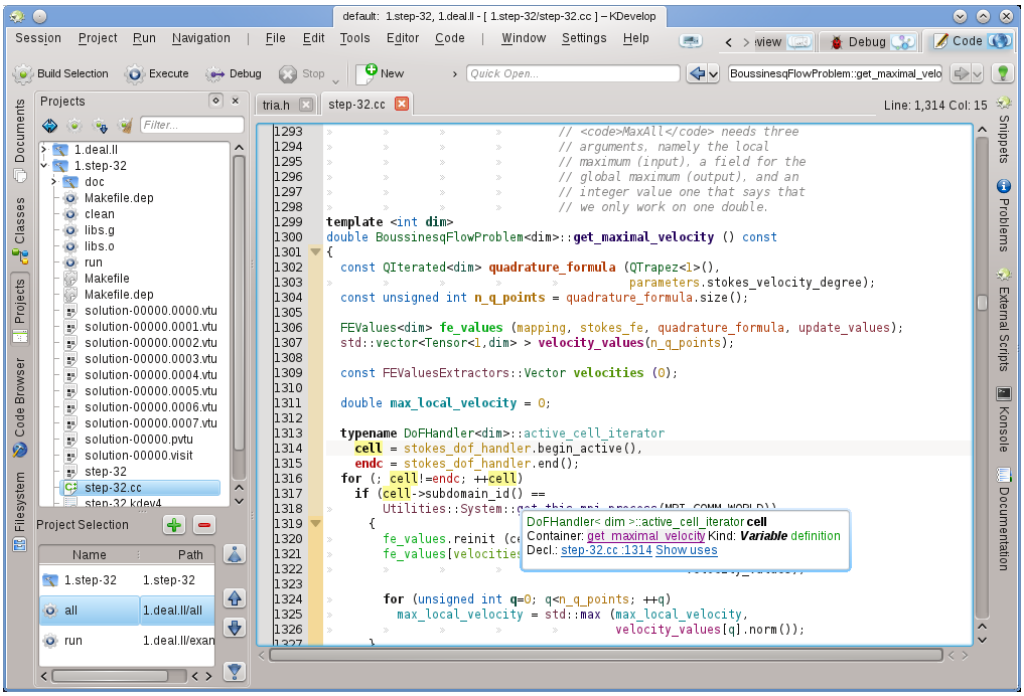
Een complete lijst van de hulpmiddelen en vensters vindt u hier.

Voor veel programmeurs is verticale monitor ruimte erg belangrijk. Om hieraan tegemoet te komen, kunt u de hulpvensters aan de linkerkant en de rechterkant van het venster arrangeren: voor het verplaatsen van een hulpmiddel klikt u erop met de rechtermuisknop en versleept het vervolgens naar de nieuwe locatie.

3.2 Broncode verkennen

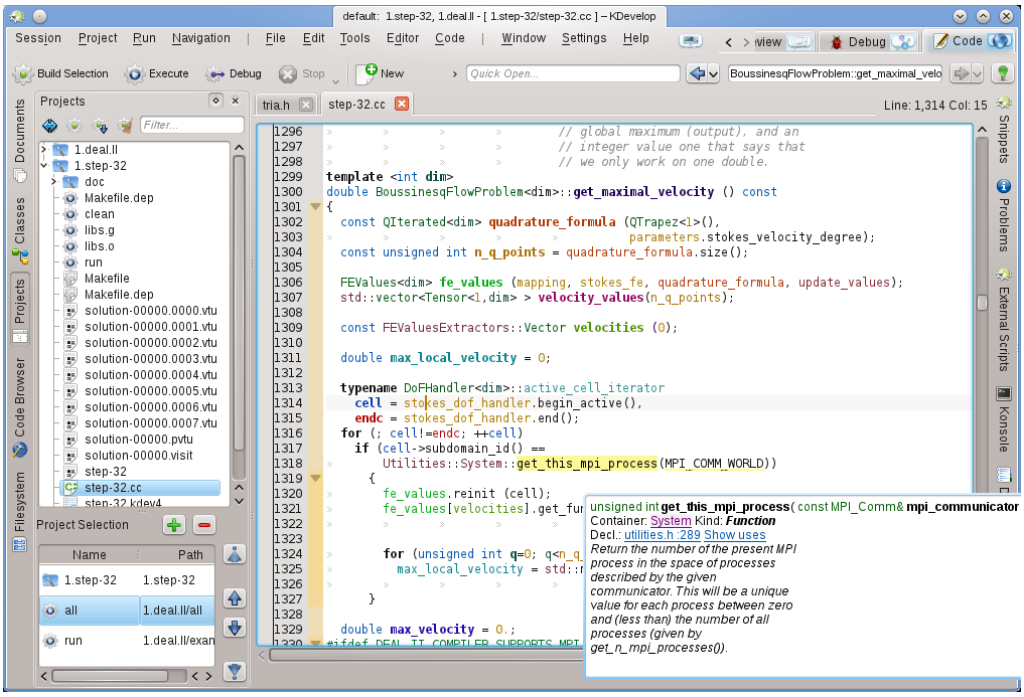
3.2.1 Lokale informatie

KDevelop *begrijpt* broncode, en als consequentie daarvan is het erg goed in het geven van informatie over variabelen en functies die in uw programma beschikbaar zijn. Hier is bijvoorbeeld een schermafdruk van het bewerken van een stuk code waarbij de muis zweeft boven het symbool `cell` in regel 1316 (als u gewent bent om met een toetsenbord te werken, u kunt hetzelfde effect bereiken door de **Alt**-toets een tijdje ingedrukt te houden):



KDevelop toont aan mij een tooltip met daarin het type variabele (hier: `DoFHandler<dim>active_cell_iterator`), waar deze variabele is gedeclareerd (de *container*, omdat die hier de surrounding functie `get_maximal_velocity` is, daar het een lokale variabele is), wat het is (een variabele, niet een functie, klasse of namespace) en waar het is gedeclareerd (in regel 1314, een paar regels hoger in de code).

In dit voorbeeld is het symbool waarboven de muis zweeft niet gedocumenteerd. Als de muis boven het symbool `get_this_mpi_process` in regel 1318 had gezweefd dan was de uitkomst dit geweest:

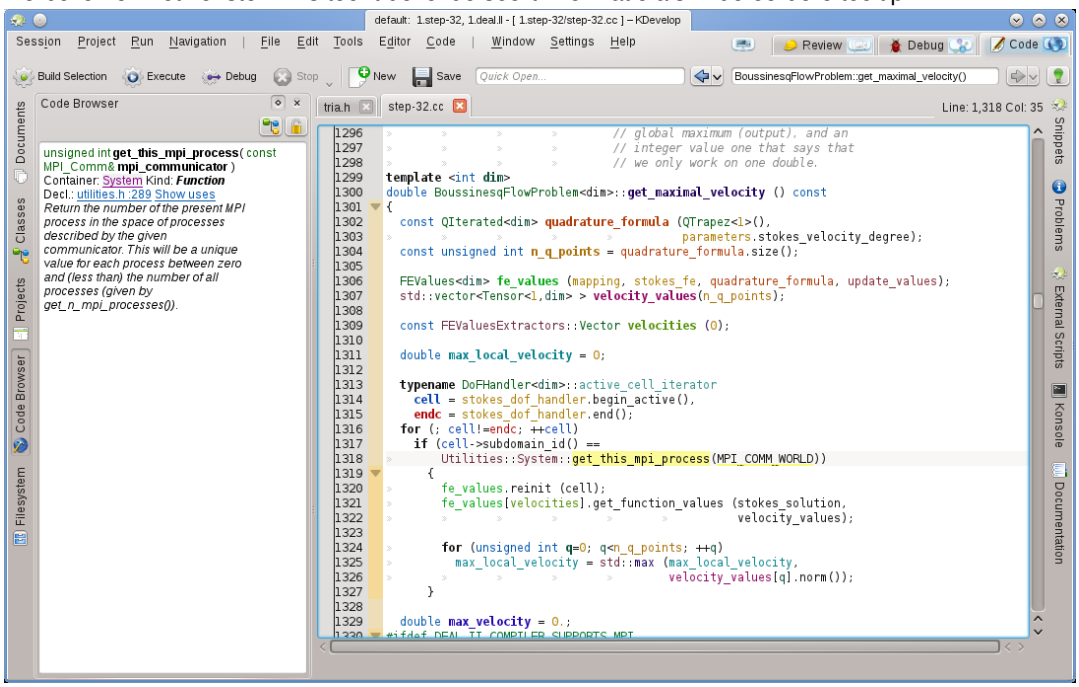


Hier heeft KDevelop een kruisverwijzing naar een declaratie uit een compleet ander bestand (`utilities.h`, die zelfs uit een compleet ander project van dezelfde sessie komt) samen met de doxygen-style commentaar die hier de declaratie begeleid.

Wat deze tooltips nog meer nuttig maakt is dat ze dynamisch zijn: ik kan erop klikken zodat ik informatie krijg over de context waarin de variabele is gedeclareerd (bijv. over de namespace `System`, waar het is gedeclareerd, gedefinieerd, gebruikt, of over de documentatie daarvan) en ik kan op de blauwe links klikken zodat de cursor naar de locatie van de declaratie van het symbool springt (bijv. in `utilities.h`, regel 289) of het geeft mij een lijst van de locaties waar dit symbool is gebruikt in het huidige bestand (of in alle projecten van deze sessie. Dit laatste is vaak nuttig als u wilt bestuderen hoe, bijvoorbeeld, een bepaalde functie in een groot stuk code is gebruikt.

OPMERKING

De informatie in een tooltip is veranderlijk — het is afhankelijk van of u de **Alt**-toets indrukt of waar u uw muis boven zweeft. Als u het op een meer permanente plaats wilt lezen, dan kunt u het subvenster **Broncode Browser** openen. Hier is bijvoorbeeld de cursor bij dezelfde functie als in voorbeeld hierboven en het venster links toont dezelfde soort informatie als in de eerdere tooltip:



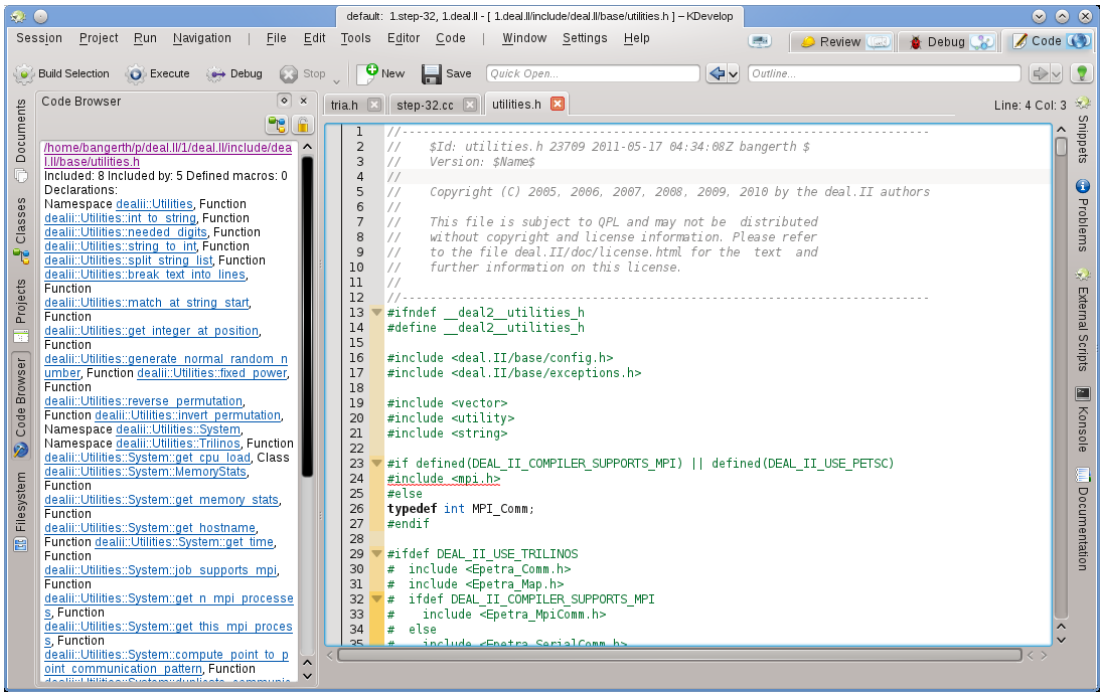
Als u de cursor rechts beweegt dan wijzigt de informatie aan de linkerkant. Maar als u klikt op de knop **Huidige weergave vergrendelen** rechtsboven dan vergrendelt u deze informatie, zodat het onafhankelijk is van de muis-bewegingen en u deze informatie rustig kunt bestuderen.

OPMERKING

Dit soort van contextinformatie is beschikbaar op veel andere plaatsen in KDevelop, niet alleen in de broncodebewerker. Bijvoorbeeld, het ingedrukt houden van de toets **Alt** in een lijst voor aanvullen(bijv. bij een snel-openen) geeft ook de contextinformatie over het huidige symbool.

3.2.2 Informatie op bestand-niveau

Een niveau hoger is het verkrijgen van informatie over het bronbestand waar u op dit moment aan werkt. Om deze informatie te verkrijgen plaatst u de cursor op het huidige bestand waarna u kunt de informatie kunt bekijken in de **Code Browser**:

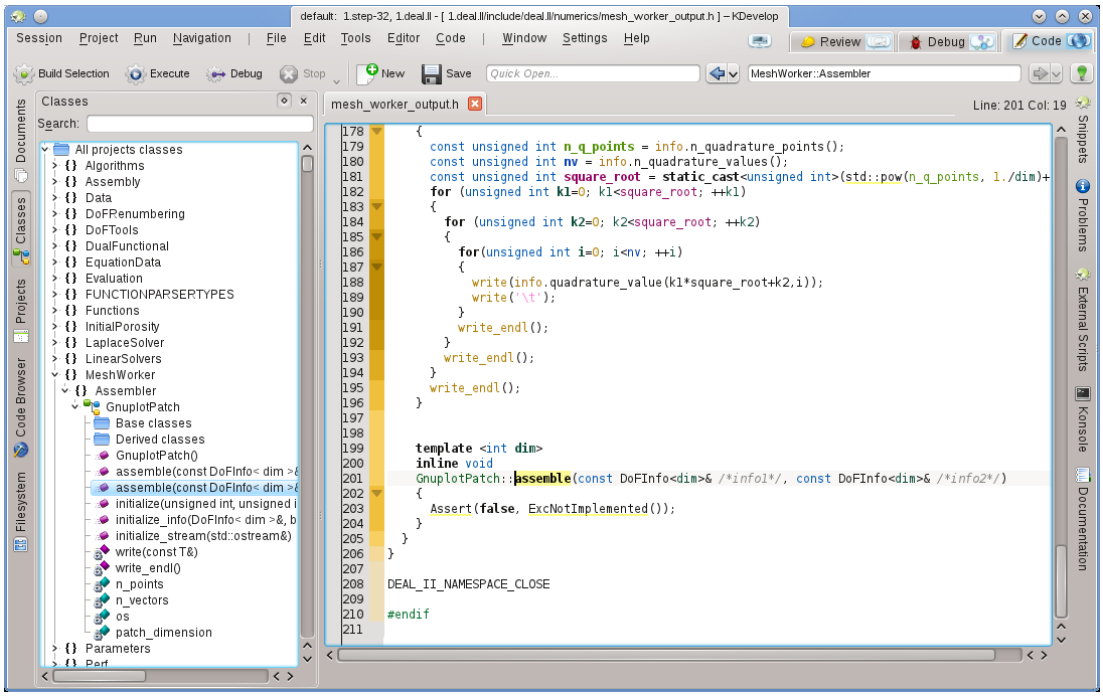


Hier toont het een lijst met namespaces, klasses en functies die gedeclareerd of gedefinieerd zijn in het geselecteerde bestand, zodat u een indruk krijgt van wat er in het bestand gebeurt en u naar elk van deze declaraties of definities kunt springen zonder in het bestand omhoog of omlaag te scrollen of te moeten zoeken naar een bepaald symbool.

> OPMERKING
> De informatie over het bestand is dezelfde als in de 'Overzicht' modus getoond zoals hieronder in de Navigeren door broncode beschreven; het verschil is dat de overzicht modus alleen een tijdelijke tooltip is.

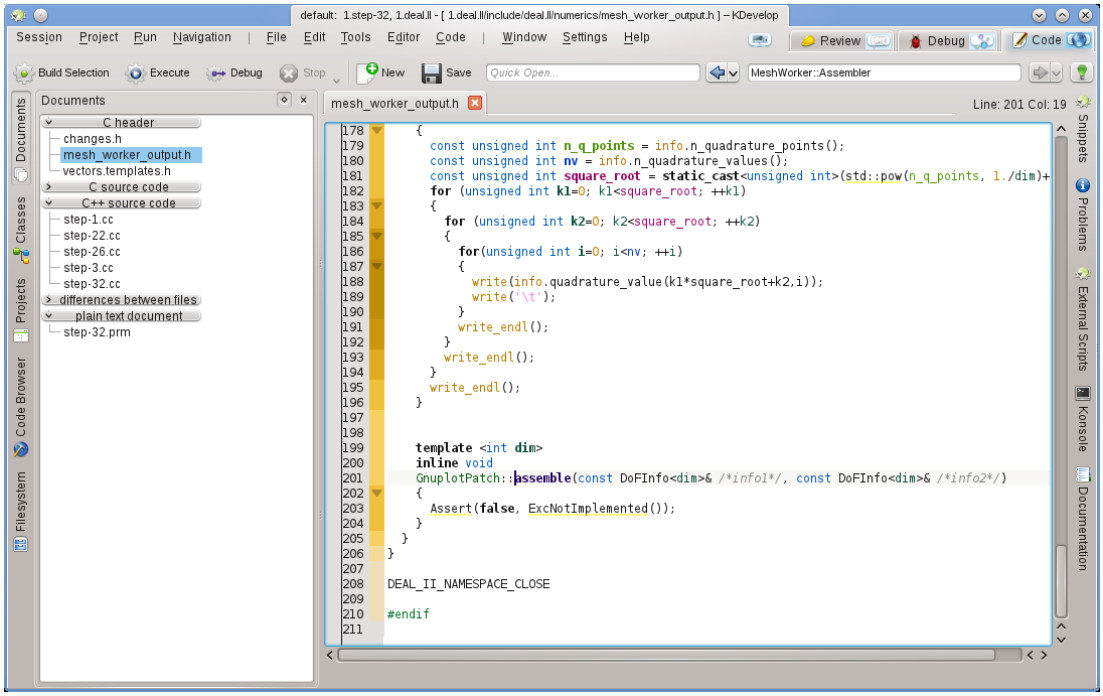
3.2.3 Informatie op project- en sessie-niveau

Er zijn zoveel manieren waarop u informatie kan verkrijgen over een heel project (of, in feite over alle projecten in een sessie). Deze soort informatie krijgt u via verschillende vensters. Bijvoorbeeld het venster **Klassen** laat voor alle projecten in de sessie een boomstructuur zien met alle klassen en bijbehorende namespaces, samen met de leden van de functies en de variabelen van elk van deze klassen:



Met de muis zwevend boven een entry krijgt u informatie over het symbool, waar het is gedeclareerd en gedefinieerd en waar het is gebruikt. Als u dubbelklikt op een entry in de boomstructuur dan opent op de locatie waar het symbool is gedeclareerd of gedefinieerd een bewerking-venster.

Maar er zijn ook andere manieren om algemene informatie te bestuderen. Bijvoorbeeld, het venster **Documenten** geeft een overzicht over een project in de vorm van de soorten bestanden of andere documenten waaruit het project is opgebouwd:



3.2.4 Veelkleurig oplichten uitgelegd

KDevelop gebruikt veel verschillende kleuren om verschillende objecten op te laten lichten in de broncode. Als u de betekenis van de verschillende kleuren weet dan kunt snel veel informatie uit de broncode halen door alleen maar naar de kleuren te kijken,zonder een enkele karakter te lezen. Het oplichten volgt de volgende regels:

- Objects van het type Class / Struct, Enum (de waarde en het type), (global) functies, en leden van klasses hebben elk een eigen kleur toegewezen (klasses zijn groen, enums zijn donkerrood, en members zijn donkergeel of paars, (algemeen) functies zijn altijd paars).
- Alle algemene variabelen hebben de kleur donkergroen.
- Identifiers die typedefs zijn voor een ander type hebben de kleur cyaan.
- Alle object-declaraties en definities zijn vetgedrukt.
- Als u een member aanroept vanuit de context waar het is gedefinieerd (base of afgeleide klasse) dan verschijnt het in geel, in de andere gevallen verschijnt het in paars.
- Als een member private of protected is dan krijgt het bij gebruik een donkerder kleur.
- Voor lokale variabelen die bij een functie horen worden alle kleuren uit de regenboog gebruikt, waarbij de kleurkeuze gebaseerd is op de hash van de identifier. Hier horen ook de parameters van de functie bij. Een identifier zal als het niet van scope verandert altijd dezelfde kleur gebruiken (maar dezelfde identifier zal een andere kleur krijgen als het een ander object vertegenwoordigt, bijv. als het wordt geherdefinieerd in een meer geneste scope), en u krijgt meestal dezelfde kleur voor dezelfde identifier in een andere scope. Als u dus meerdere functies heeft waarbij de parameters dezelfde namen hebben dan lijken de parameters qua kleur op elkaar. U kunt deze markering-kleuren apart uitschakelen in het instellingendialoog.
- Identifiers waar KDevelop niet de overeenkomstige declaratie van kon bepalen worden wit gekleurd. Dit kan soms veroorzaakt worden door het ontbreken van een `#include` directive.
- Naast deze kleur-markering is ook de normale syntax-markering van toepassing zoals we dat kennen van Kate. Bij een conflict heeft KDevelop's semantische markering altijd de voorrang boven de markering van de editor.

3.3 Navigeren door broncode

In het vorige hoofdstuk hebben we het doorzoeken van broncode beschreven, bijv. het verkrijgen van informatie over symbols, bestanden en projecten. De volgende stap is het in uw broncode rondspringen, bijv. er in navigeren. Opnieuw is dit op verschillende niveaus mogelijk: lokaal, in een bestand en in een project.

> OPMERKING
> Vanuit het menu **Navigatie** in het hoofdvenster van KDevelop zijn veel verschillende manieren beschikbaar om door de broncode te navigeren.

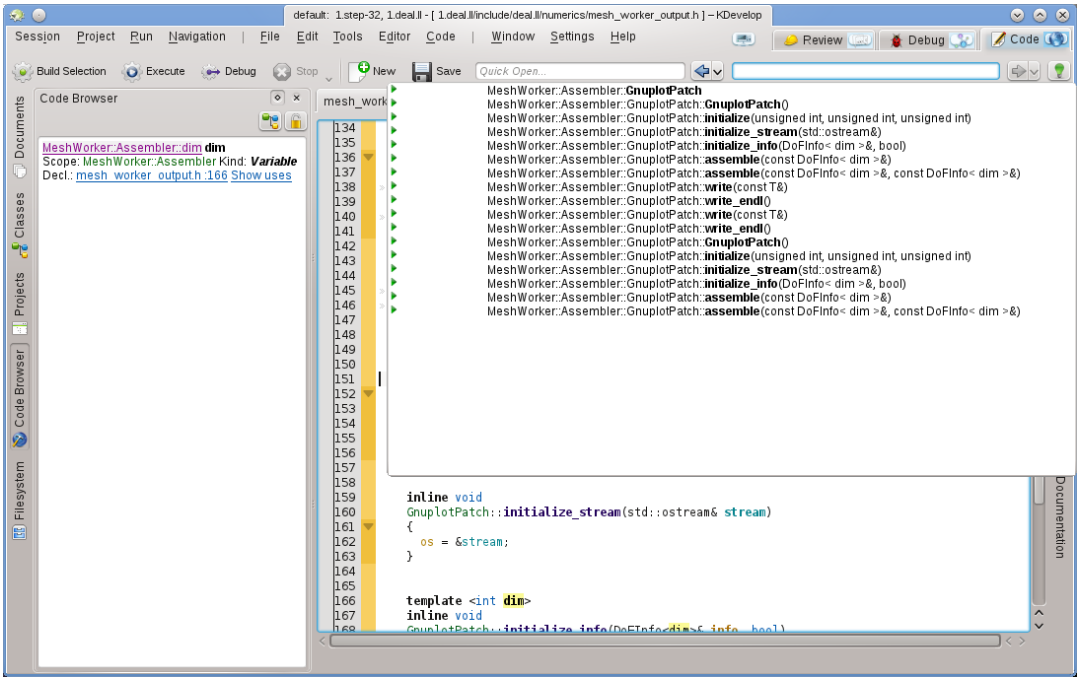
3.3.1 Lokale navigatie

KDevelop is veel meer dan een editor, maar het is *ook* een editor voor broncode. Bij gebruik als zodanig kunt u natuurlijk in een bronbestand de cursor omhoog, omlaag, naar links of naar rechts verplaatsen. U kunt ook de **vorig scherm** en **volgende scherm** toetsen gebruiken, en alle andere commando's die u kent uit andere editors.

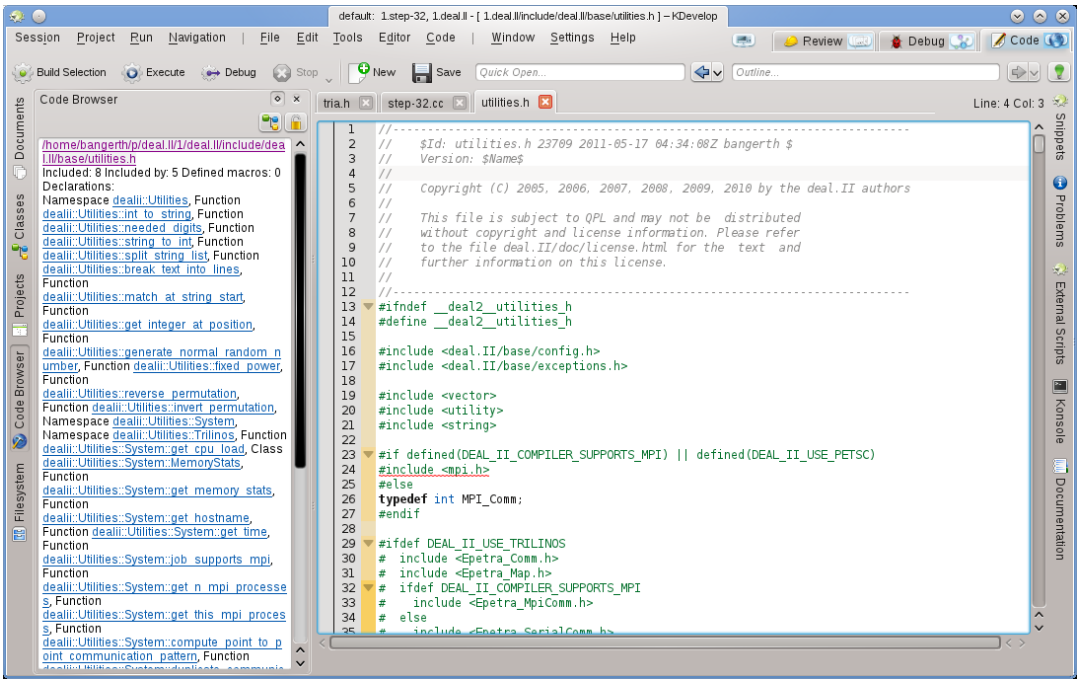
3.3.2 Overzicht en navigatie op bestandsniveau

KDevelop heeft op bestandsniveau veel verschillende manieren om door de broncode te navigeren. Als voorbeeld:

- **Uitlijning**: U kunt op minstens drie verschillende manieren een overzicht van het huidige bestand verkrijgen:
  - Klik in het tekstvak van **Uitlijning** rechtsboven van het hoofdvenster of toets in **Alt-Ctrl-N** om een keuzelijst met alle functies en klasse-declaraties te openen:



    U kunt vervolgens een keuze maken welke u wilt selecteren, of — als de keuze groot is — voer een tekst in wat vervolgens zal verschijnen in het tekstvak; als u doorgaat met het invoeren van tekst zal de lijst korter en korter worden omdat namen afvallen die niet overeenkomen met de al ingevoerde tekst totdat u tenslotte een keuze maakt.
  - Plaats uw cursor op het bestand-niveau (bijv. buiten elke functie of klassedeclaratie of definities) en houdt het **Broncode Browser**-venster geopend:



Dit geeft u ook een overzicht van wat er in het geselecteerde bestand gebeurt, en hierdoor kunt u ook makkelijk selecteren waarnaar u naar toe wilt springen.

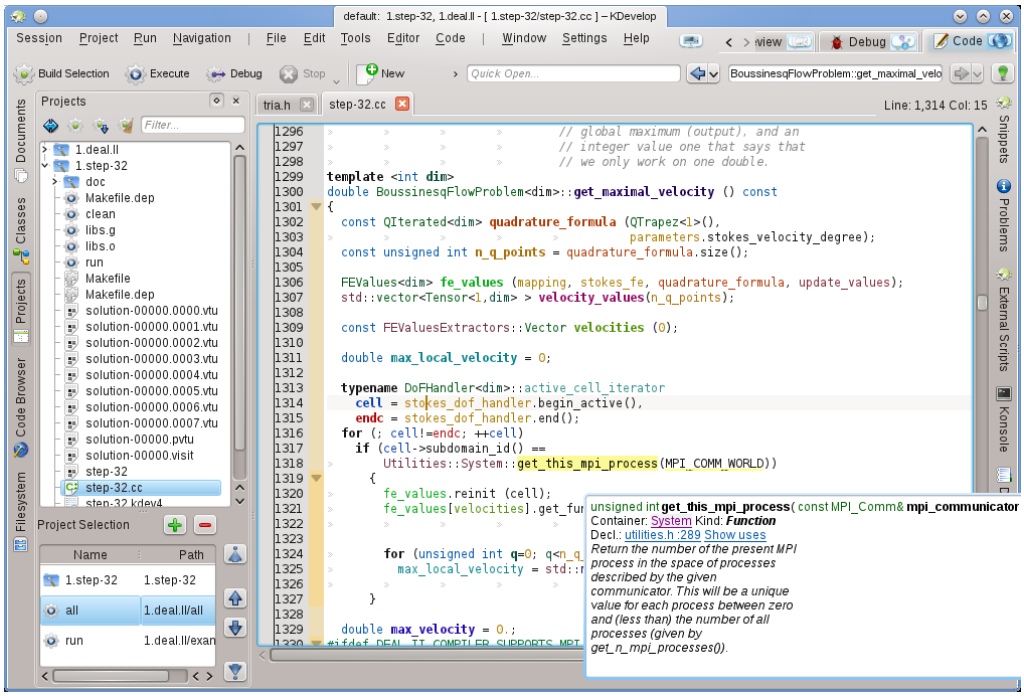
  - Als uw muis boven de tab van een geopend bestand zweeft dan krijgt u ook een overzicht van het bestand in die tab.

- Bronbestanden zijn georganiseerd als een lijst met functiedeclaraties of definities. Toets in **Alt-Ctrl-PgUp** en **Alt-Ctrl-PgDown** om naar de vorige of volgende definitie in het bestand te gaan.

3.3.3 Navigeren in projecten en sessies: Semantische navigatie

Zoals ook op andere plaatsen gemeld, KDevelop kijkt niet zozeer naar aparte bronbestanden maar meer naar projecten als een geheel (of eigenlijk naar alle projecten die deel uitmaken van de huidige sessie). Als consequentie daarvan heeft het veel mogelijkheden om door complete projecten te navigeren. Sommige zijn al eerder beschreven in het hoofdstuk Broncode verkennen terwijl andere compleet anders zijn. De overeenkomst is dat deze navigatie mogelijkheden gebaseerd zijn op een *begrip van de betekenis* van de code, bijv. zij bieden iets waarbij het nodig is dat het complete project ontleedt wordt en de data met elkaar verbonden wordt. De volgende lijst toont enkele manieren hoe u door broncode kan navigeren die verspreidt is door een mogelijk groot aantal bestanden:

- Zoals te lezen in het hoofdstuk over Broncode verkennen, u krijgt een tooltip waarin u de individuele namespace, klasse, functie of de naam van de variabele te zien krijgt als u met de muis erboven zweeft of door de **Alt**-toets enige tijd indrukt. Hier ziet u een voorbeeld:



Als u klikt op de een declaratie van een symbool of de lijst met locaties uitklapt kunt u naar een van deze locaties springen waarbij indien noodzakelijk het bestand wordt geopend waarna de cursor op de locatie wordt geplaatst. Een vergelijkbaar resultaat krijgt u als u zoals eerder beschreven de **Broncode Browser** gebruikt.

- Een snellere manier om naar de declaratie van een symbool te springen zonder te hoeven klikken op de koppelingen in de tekstballon is om tijdelijk **Modus broncode browsen** in te schakelen door de toets **Alt** of **Ctrl** ingedrukt te houden. In deze modus is het mogelijk om direct op een symbool in de editor te klikken om naar zijn declaratie te springen.

- **Snel openen**: een zeer handige manier om snel naar andere bestanden of locaties te gaan is door het gebruik van de verschillende *snel open* methoden in KDevelop. Er zijn vier versies hiervan:
  - **Snel openen van klasse** (**Navigatie** → **Snel openen van klasse** of **Alt-Ctrl-C**): U krijgt vervolgens een lijst met alle klassen in de projecten van deze sessie. Start met het typen van (een gedeelte van) de naam van een klas en lijst zal inkrimpen tot alleen die overblijven die overeenkomen met wat u tot dusver heeft ingevoerd. Als de lijst kort genoeg is dan kan u een element selecteren door gebruik van de up en down toetsen en KDevelop brengt u naar de plaats waar de klasse is gedeclareerd.
  - **Functie snel openen** (**Navigatie** → **Functie snel openen** of **Alt-Ctrl-M**): U krijgt vervolgens een lijst met alle (lid)functies in de projecten van deze sessie en u kunt ze op dezelfde manier selecteren als hierboven beschreven. Merk op dat deze lijst zowel functiedeclaraties als functiedefinities aanwezig kunnen zijn.
  - **Snel openen van bestand** (**Navigatie** → **Snel openen van bestand** of **Alt-Ctrl-O**): U krijgt vervolgens een lijst met alle bestanden in de projecten van deze sessie en u kunt ze op dezelfde manier selecteren als hierboven beschreven.
  - **Universeel snel openen** (**Navigatie** → **Snel openen** of **Alt-Ctrl-Q**): Als u vergeten bent welke toetscombinatie bij welke van bovengenoemde commando's hoort, deze is het universele Zwitserse zakmes — het toont u een gecombineerde lijst met alle bestanden, functies, klasses en andere zaken waaruit u kunt selecteren.

- **Ga naar declaratie/definitie**: Wanneer men een (member) functie implementeert dan moet men vaak terug gaan naar de locatie waar de functie is declareert om bijvoorbeeld de lijst met functie argumenten bij de declaratie en de definitie synchroon te houden of om de documentatie bij te werken. Om dit te doen moet u de cursor op de naam van de functie plaatsen en

vervolgens **Navigatie** → **Ga naar declaratie** (of toets in **Ctrl-.**) selecteren om naar de plaats te gaan waar de functie is gedeclareerd. Er zijn meerdere manieren om terug te gaan naar het startpunt:

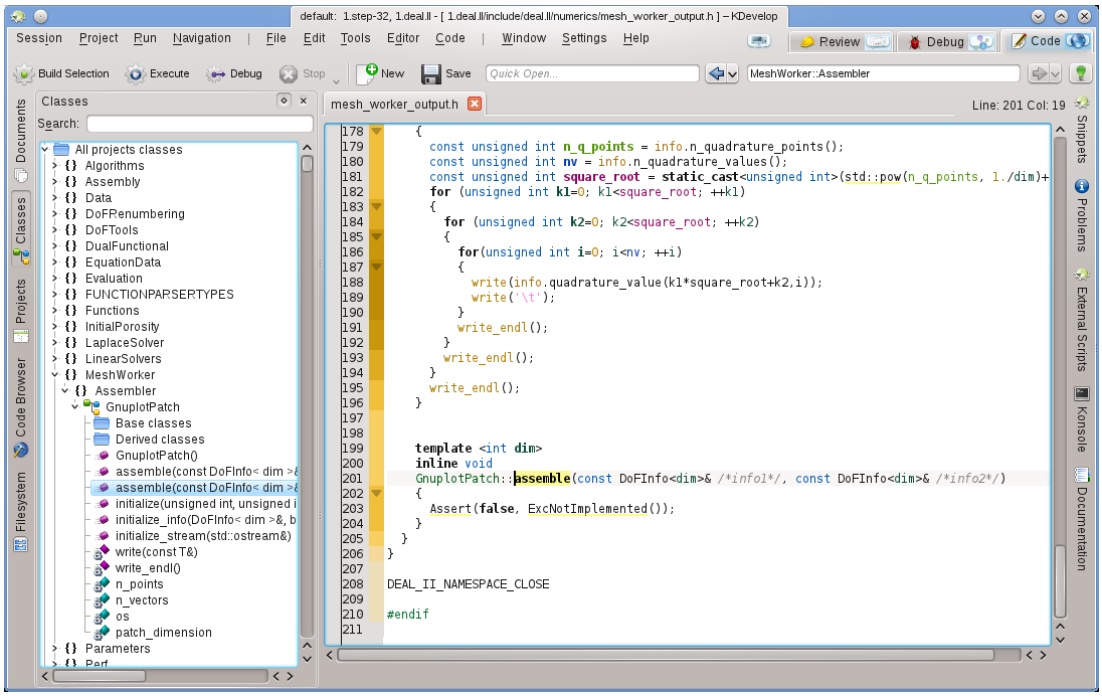
– Selecteer **Navigatie** → **Ga naar definitie** (or toets in **Ctrl-,**).
– Selecteer **Navigatie** → **Vorige bezochte context** (of toets in **Meta-Links**), zoals hieronder beschreven.

OPMERKING
Het springen naar de declaratie van een symbool is iets dat niet alleen werkt wanneer de cursor is geplaatst op de functienaam waaraan u werkt. Maar dit werkt ook bij andere symbolen: plaats de cursor op een (locale, globale, of member) variabele en springen naar zijn declaratie verplaatst u ook naar de locatie van zijn declaratie. Vergelijkbaar kan u ook de cursor op de naam van een klasse plaatsen, bijvoorbeeld in een variabele of een functiedeclaratie en vervolgens naar de locatie van de declaratie daarvan springen.

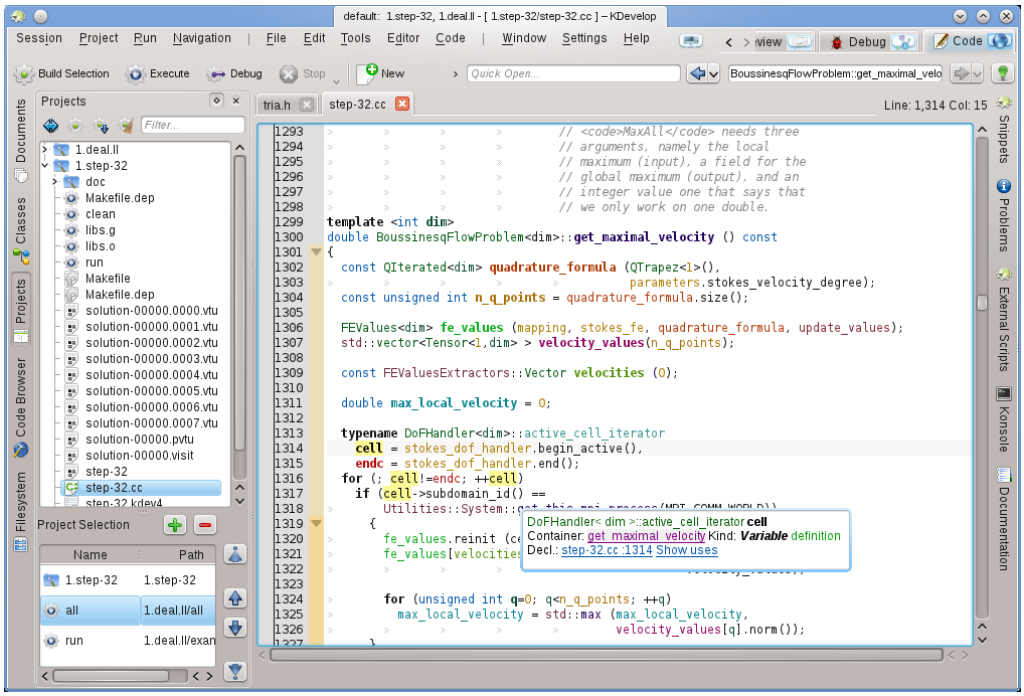
- **Schakel definitie/declaratie om**: In het voorbeeld hierboven , om naar de locatie van de declaratie van de huidige functie te springen moet u eerst de cursor op de functienaam plaatsen. U kunt ook **Navigatie** → **Schakel definitie/declaratie om** gebruiken (of toets **Shift-Ctrl-C**) om naar de declaratie te springen van de functie waar de cursor op dit moment is. Door het dezelfde menu-item opnieuw te selecteren springt u terug naar de plaats waar de functie is gedefinieerd.
- **Vorig/Volgend gebruik**: plaats de cursor op de naam van een lokale variabele en selecteer **Navigatie** → **Volgend gebruik** (of toets **Meta-Shift-Rechts**) en u springt naar de volgende plaats waar deze variabele in de code is gebruikt. (Merk op dat er niet alleen gezocht wordt naar de volgende plek waar de variabele voorkomt maar neemt ook in overweging dat variabele met dezelfde naam in andere scopes verschillend zijn.) Dit werkt op dezelfde manier bij het gebruik van functienamen. Selecteer **Navigatie** → **Vorig gebruik** (of toets **Meta-Shift-Links**) en u springt naar de vorige plaats waar deze variabele in de code is gebruikt.

OPMERKING
Voor het bekijken van een lijst met alle locaties waar deze commando's doorheen springen, plaatst u de cursor er boven op en opent het **Broncode Browser** venster of houdt de **Alt** toets ingedrukt. Dit wordt meer gedetailleerd uitgelegd in het hoofdstuk Broncode verkennen.

- De **context lijst**: Web browsers hebben deze mogelijkheid waarmee u de mogelijkheid heeft om vooruit en achteruit te gaan in de lijst met recent bezochte webpagina's. KDevelop heeft dezelfde mogelijkheid, behalve dat u in plaats van webpagina's *contexts* bezoekt. Een context is de huidige locatie van de cursor, en u wijzigt deze door naar een andere locatie te navigeren met behulp van cursor opdrachten — bijvoorbeeld, door te klikken op een locatie die u via een tooltip is aangereikt, via het **Broncode Browser**-venster, een item te kiezen in het **Navigatie** menu, of een ander navigatie-commando. Door het gebruik van de **Navigatie** → **Vorige bezochte Context** (**Meta-Links**) en **Navigatie** → **Volgende bezochte Context** (**Meta-Rechts**) gaat u door de lijst met bezochte context op dezelfde manier als u met de knoppen **vorige** en **volgende** van een browser naar de vorige of volgende webpagina in de lijst met bezochte pagina's gaat.
- Tenslotte zijn er hulpmiddelen waarmee u naar andere locaties in uw broncode kunt navigeren. Bijvoorbeeld, het hulpmiddel **Klasses** toont een lijst met alle namespaces en klasses in alle projecten van de huidige sessie, deze lijst kunt u uitklappen zodat u van elke klasse de member functies en variabelen kunt zien:



Door dubbelklikken op een item (of via het contextmenu door te klikken met de rechtermuisknop) springt u naar de locatie van de declaratie van het item. Andere hulpmiddelen werken vergelijkbaar; bijvoorbeeld, het venster **Projecten** toont een lijst met bestanden die deel uitmaken van een sessie:



Ook hier, dubbelklikken opent het bestand.

3.4 Broncode schrijven

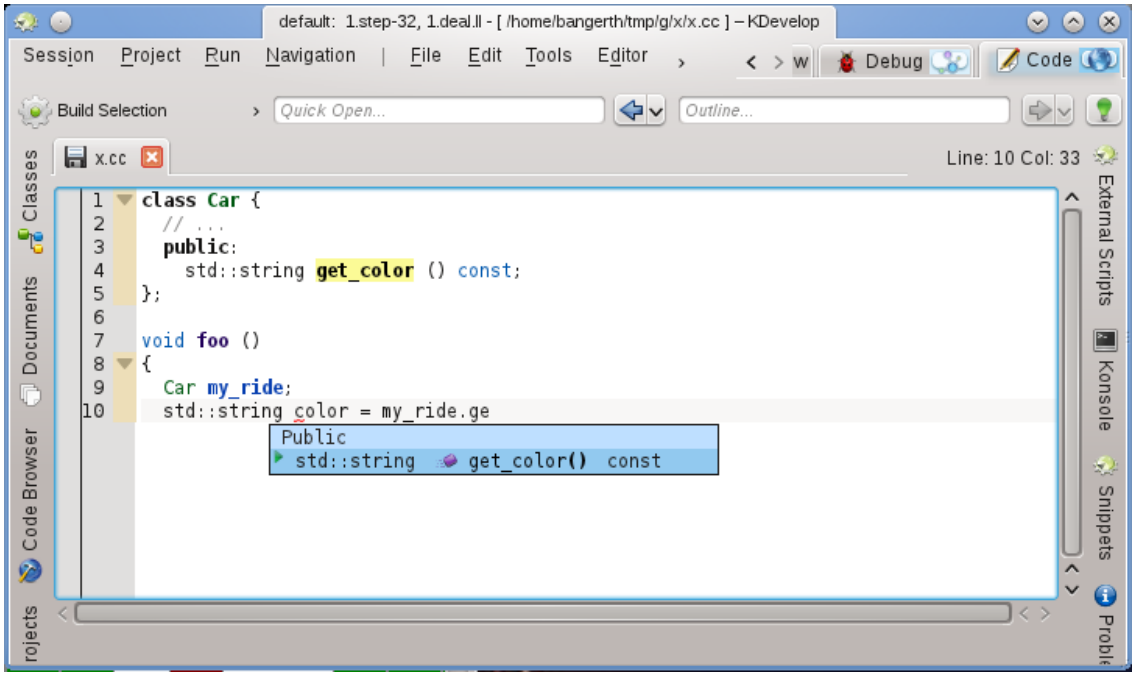
Omdat KDevelop de broncode in uw project begrijpt, kan het assisteren bij het schrijven van meer code. Hieronder zijn enige manieren beschreven waarop het dat doet.

3.4.1 Automatische aanvulling

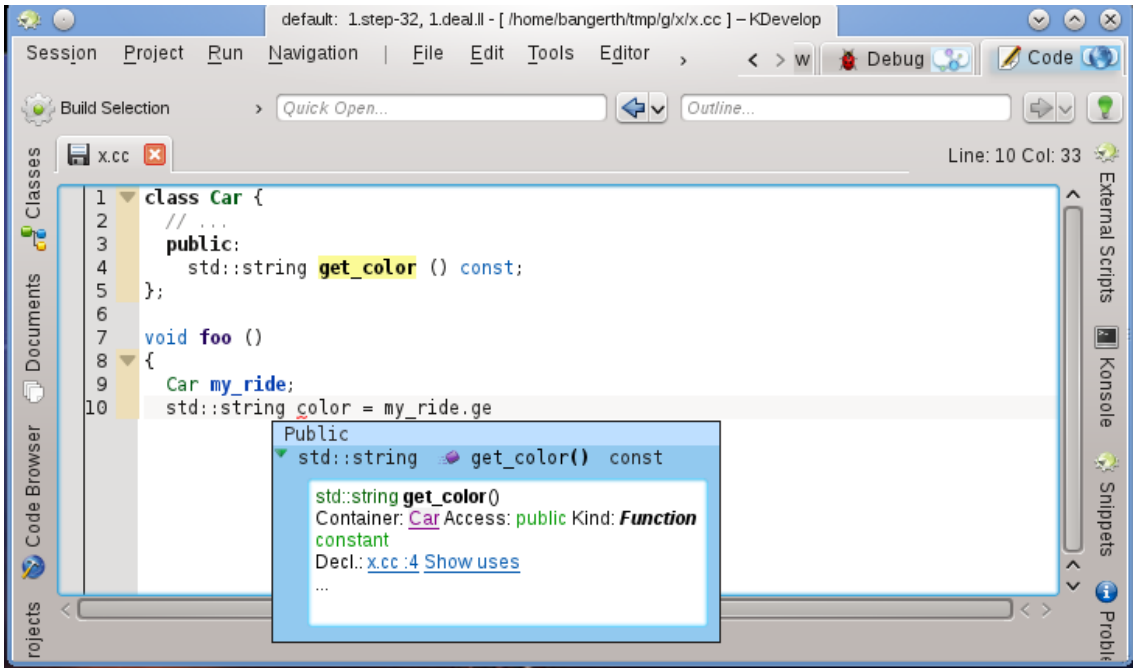
Waarschijnlijk het meest gebruikte functionaliteit bij het schrijven van nieuwe code is automatisch aanvullen. Denk bijvoorbeeld aan het volgende stuk code:

```
class Car {
  // ...
  public:
    std::string get_color () const;
};
void foo()
{
  Car my_ride;
  // ...doe iets met deze variabele...
  std::string color = my_ride.ge
```

In de laatste regel zal KDevelop weten dat de variabele `my_ride` van het type `Car` is en zal automatisch aanbieden om de naam van de member function `ge` aan te vullen tot `get_color`. U hoeft slechts door te gaan met typen tot dat de functie automatisch aanvullen het aantal matches heeft beperkt tot één waarna u de **Enter**-toets kan indrukken:



Merk op dat u op de tooltip kunt klikken om meer informatie te krijgen over de functie zoals het return type en of het public is:



Automatische aanvulling kan u veel tijd besparen als u lange namen voor variabelen en functies gebruikt; bovendien vermijdt u op deze manier spellingsfouten (en de daaruit resulterende compile errors) en het is veel eenvoudiger om de precieze namen van functies te onthouden; bijvoorbeeld, als al uw getters starten met `get_`, dan zal de automatische aanvulling u alleen een lijst met mogelijke getters tonen nadat u de eerste vier letters heeft ingevoerd, u eraan herinnerend welke van de functies de correcte is. Merk op dat om automatische aanvulling te laten werken het niet nodig is dat de declaratie van de klasse `Car` of de variabele `my_ride` in hetzelfde bestand is als waar u op dit moment code bewerkt. KDevelop hoeft alleen te weten dat deze klassen en variabelen ermee verbonden zijn , bijv. de bestanden waarin de verbindingen zijn gemaakt moeten onderdeel maken van het project waaraan u op dit moment werkt.

OPMERKING

Soms start KDevelop niet automatisch met het u assisteren van het bewerken van de code. Als de automatische aanvulling tooltip niet automatisch opent, toets dan **Ctrl**-**Spatie** om de lijst met automatische aanvulling handmatig te openen. Automatische aanvulling kan pas werken nadat KDevelop uw bronbestanden heeft doorzocht. Dit gebeurt voor alle bestanden van de projecten in deze sessie in de achtergrond nadat u KDevelop heeft gestart, maar ook als u een fractie van een seconde heeft gestopt met typen (de vertraging kunt u instellen).

OPMERKING

KDevelop doorzoekt alleen bestanden die het als broncode beschouwt, dit gebeurt aan de hand van het MIME-type van het bestand. Dit type wordt pas ingesteld bij het voor de eerste keer opslaan van het bestand; als consequentie daarvan werkt de automatisch aanvulling niet als u een bestand heeft gestart en hierin code schrijft totdat het voor de eerste keer is opgeslagen.

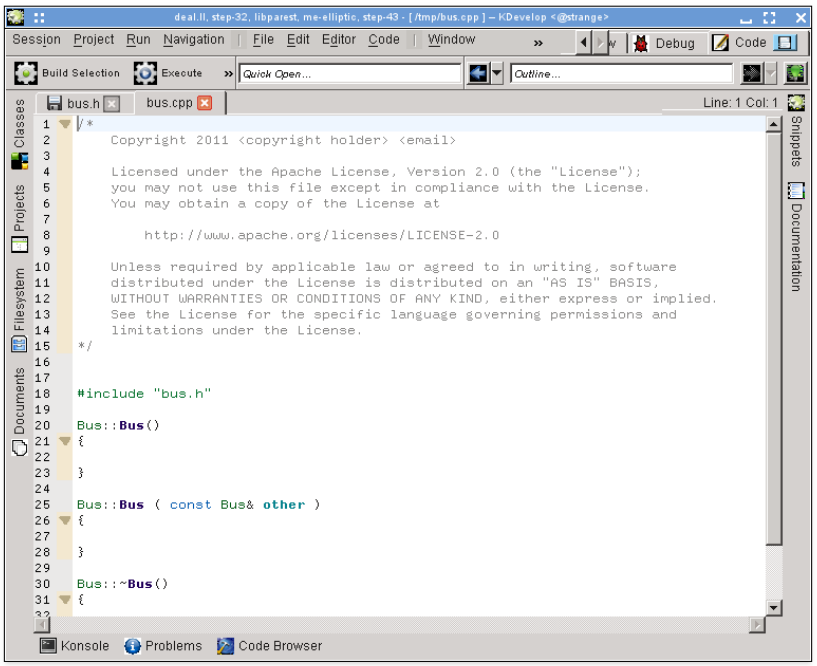
> **OPMERKING**
> Zoals eerder opgemerkt, om automatische aanvulling te kunnen laten werken moet het voor KDevelop mogelijk zijn om de declaraties te kunnen vinden in de header-bestanden. Hiervoor doorzoekt het een aantal standaard locaties. Als het geen header-bestand kan vinden dan onderstreept het met rood de naam van een header-bestand; klik daarop in dat geval om KDevelop duidelijk te maken waar het deze bestanden kan vinden en welke informatie deze geven.

> **OPMERKING**
> De beschrijving van het instellen van de automatische aanvulling vindt u in dit hoofdstuk van deze handleiding.

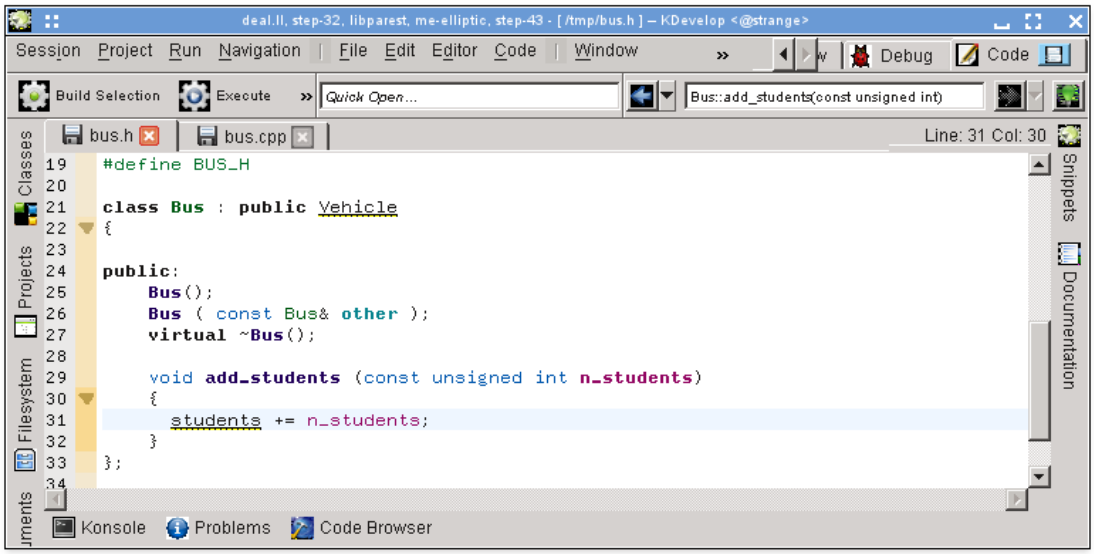
3.4.2 Nieuwe klassen toevoegen en implementeren van member-functies

KDevelop heeft een hulpmiddel voor het toevoegen van nieuwe klasses. De beschrijving van de procedure vindt u in Een nieuwe klasse aanmaken. U kunt een eenvoudige C++ klasse aanmaken door gebruik van de Basic C++ template uit de `Class` category. In het hulpmiddel kunnen we voorgedefinieerde member functies uitkiezen zoals bijvoorbeeld een lege constructor, een copy constructor of een destructor.

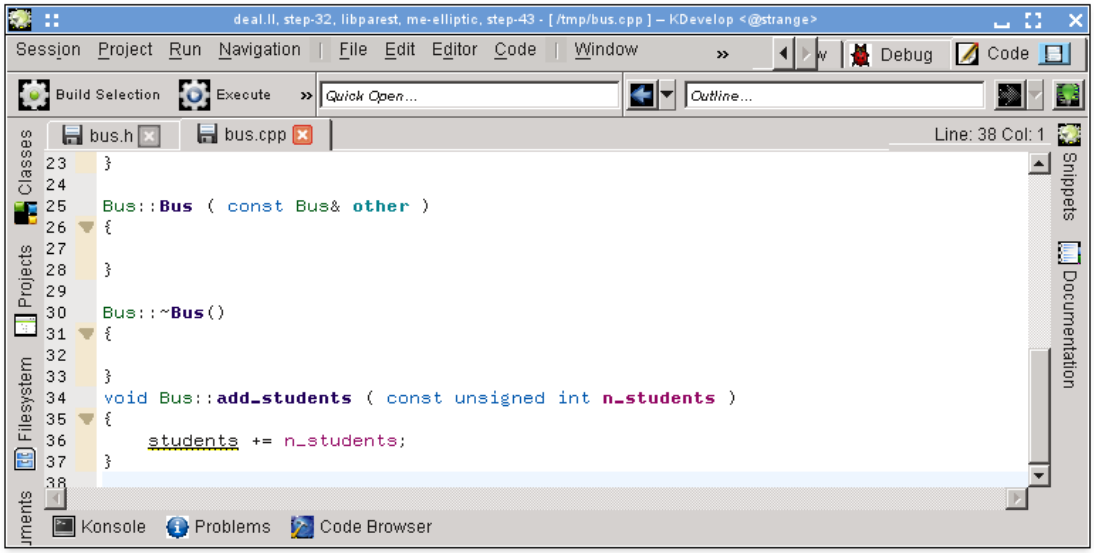
Na het gebruik van het hulpmiddel zijn de nieuwe bestanden aangemaakt en de editor geopend. Het header-bestand is inclusief guards en de nieuwe klasse heeft alle geselecteerde functies. De volgende twee stappen zijn het documenteren van de klasse en zijn memberfuncties en het implementeren daarvan. Wij zullen de hulpmiddelen voor het documenteren van de klassen en functies hieronder beschrijven. Voor het implementeren van de al aangemaakte speciale functies gaat u naar de tab **bus.cpp** waar het skelet van de functies al wacht:



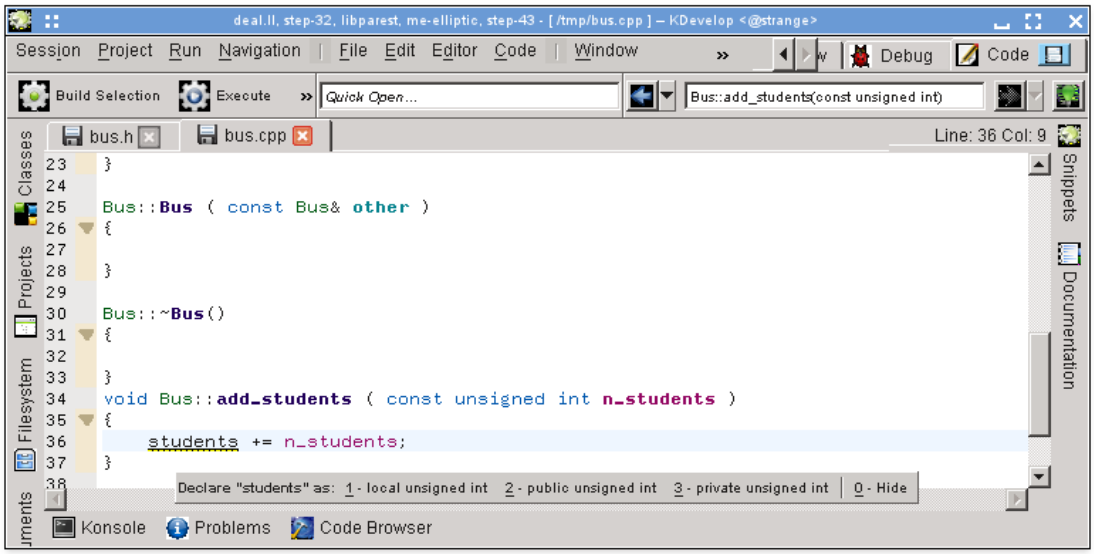
Als u nieuwe member functies wilt toevoegen dan gaat u opnieuw naar de tab **bus.h** en voegt vervolgens de naam van een functie toe. Wij willen bijvoorbeeld het volgende toevoegen:



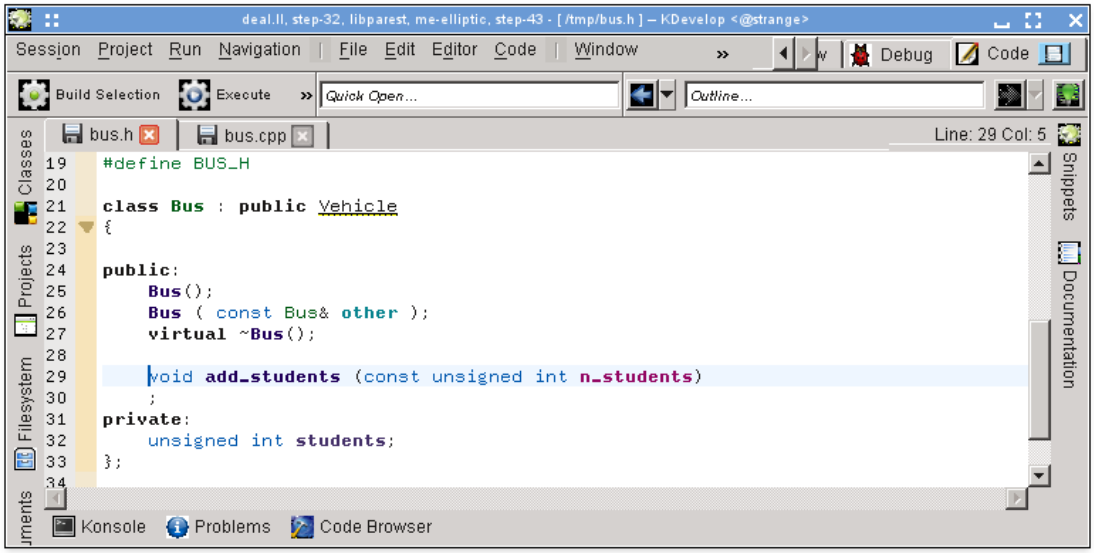
U merkt dat ik al gestart ben met met de implementatie. Maar in veel codeerstylen is het niet toegestaan dat u de functie implementeert in het headerbestand, in plaats daarvan moet dat gebeuren in het bijbehorende .cpp-bestand. Plaats de cursor op de naam van de functie en selecteer **Code** → **Verplaatsen naar bron** of toets **Ctrl-Alt-S**. Dit verplaatst de code tussen de accolades uit het headerbestand (en vervangt dit door een puntkomma om de functiedefinities af te sluiten) naar het bronbestand:



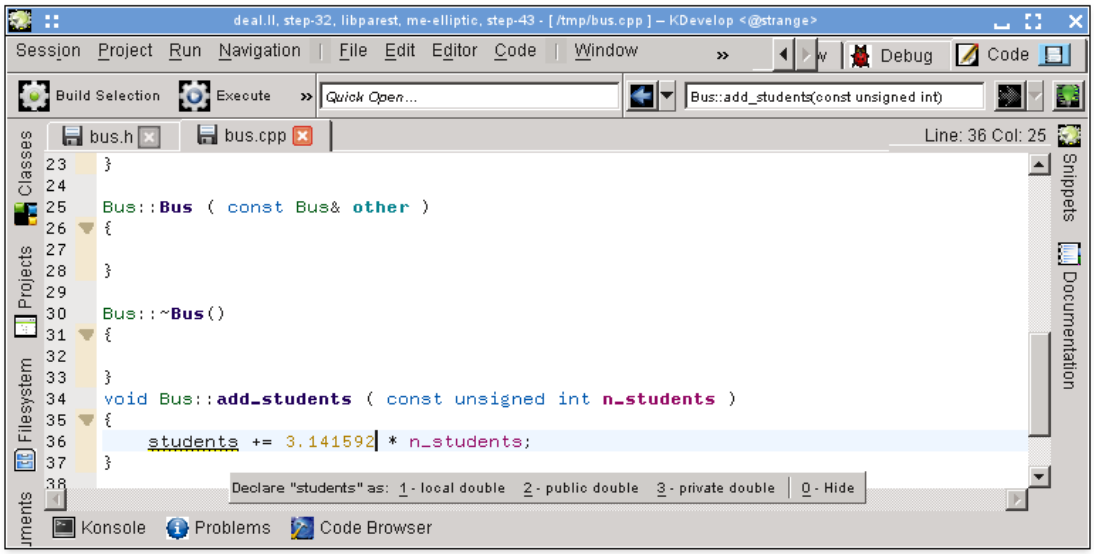
U merkt op dat ik gewoon gestart ben met typen en dat het mijn bedoeling is dat de variabele `students` waarschijnlijk een member variabele van de klasse `Bus` wordt maar dat ik het nu nog niet heb toegevoegd. Merk ook op dat KDevelop het onderstreept omdat er verder nog geen informatie over aanwezig is. Maar dat probleem kunt u oplossen: als u klikt op de naam van de variabele krijgt u de volgende tooltip:



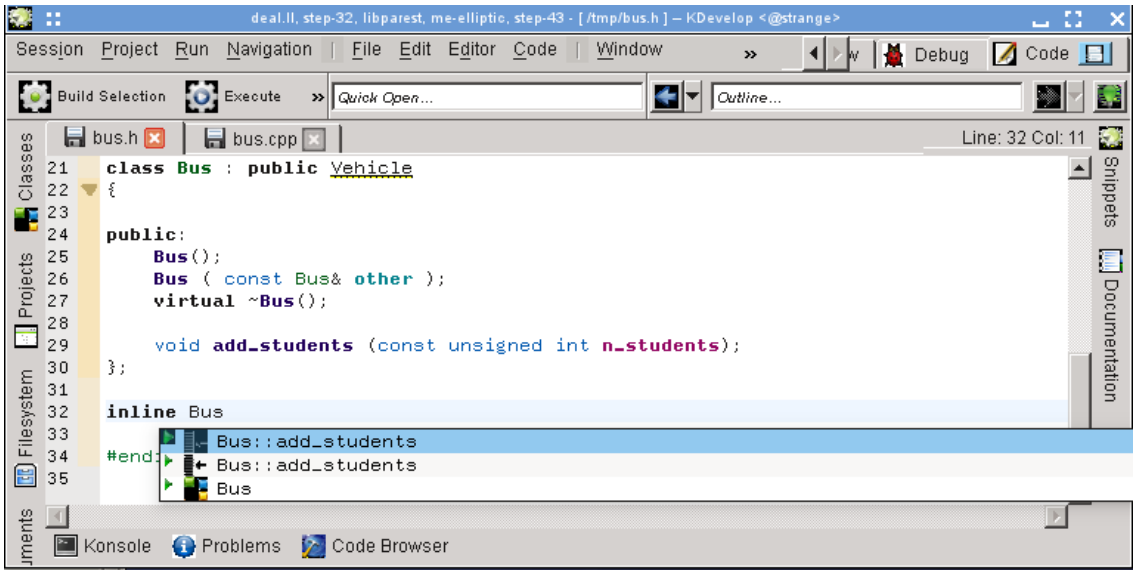
(Hetzelfde kan bereikt worden door er rechts op te klikken en **Oplossing: Declareren als** te selecteren.) Laat me '3 - private unsigned int' selecteren (ofwel met de muis of door **Alt-3**) in te drukken en kijk dan hoe het uit het header-bestand komt:



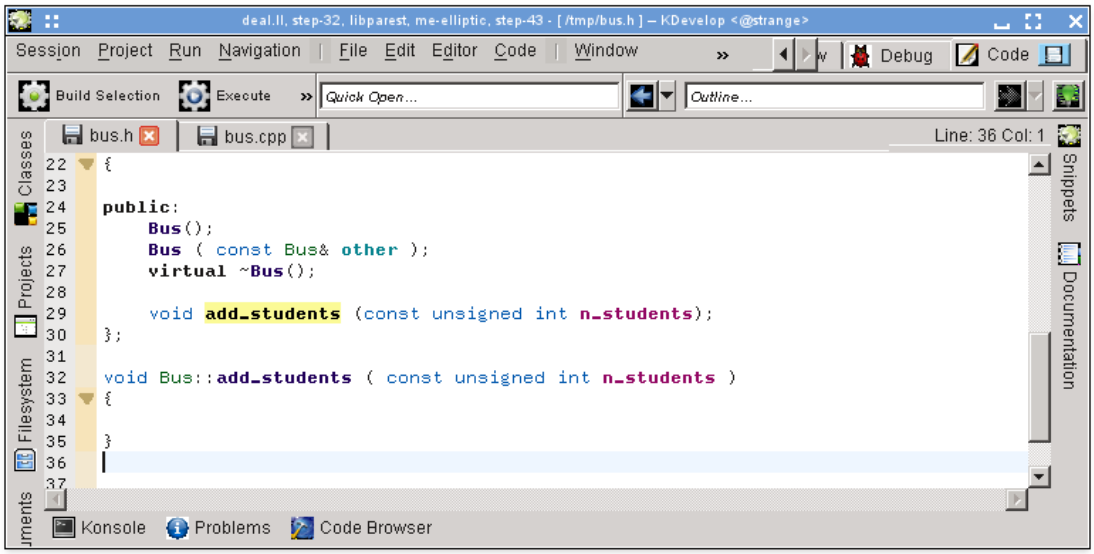
Het is het waard op te merken dat KDevelop het type van de variabele voor de declaratie haalt uit de expressie gebruikt om deze te initialiseren. Als we bijvoorbeeld de optelling hadden geschreven op de volgende tamelijk dubieuze manier, dan zou het hebben gesuggereerd om de variabele als type `double` te declareren:



Een laatste opmerking: De gebruikte methode **Code** → **Verplaatsen naar bron** voegt de nieuwe member functie niet altijd op de gewenste locatie in. U wilt het bijvoorbeeld markeren als `inline` en het onderaan het headerbestand plaatsen. In dit soort gevallen schrijft u de declaratie en start het schrijven van de functie op deze manier:



KDevelop geeft automatisch alle mogelijke aanvullingen voor wat hier zou kunnen komen. Door het selecteren van een van de twee `add_students` entries krijgt u de volgende code met al de complete lijst met argumenten ingevuld:




```
{

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 )
{

}
```

OPMERKING

Het accepteren van een van de keuzes in het voorbeeld geeft na het automatisch aanvullen de correcte ondertekening maar verwijdert helaas de al eerder geschreven `inline`-bladwijzer. Dit is gemeld als KDevelop Bug 274245.

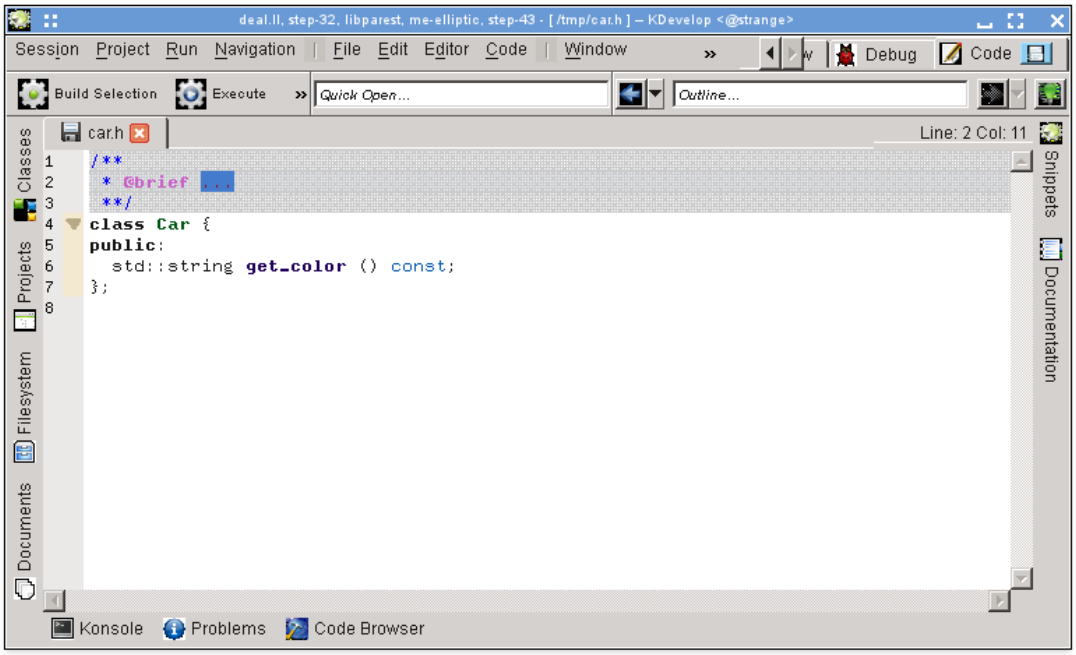
3.4.3 Documenteren van declaraties

Goede code is goed gedocumenteerd, zowel op het niveau van het implementeren van de algoritmes in de functies als op het niveau van de interface — bijv., klasses, (member en globaal) functies, en (member of globaal) variabelen moeten gedocumenteerd zijn wat betreft hun doel, mogelijke waarden voor de argumenten, pre- en postconditions, etc. Wat betreft de documentatie van de interface, doxygen is de facto standaard geworden voor de opmaak van commentaar zodat het geëxtraheerd en daarna op doorzoekbare webpagina's getoond kan worden.

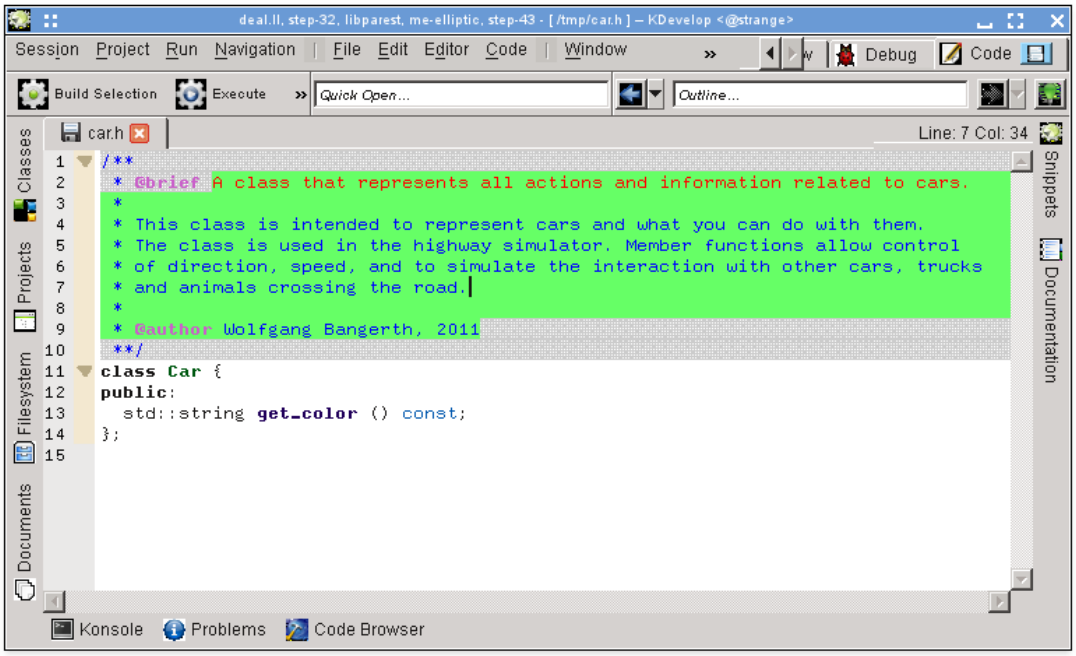
KDevelop ondersteund deze stijl van commentaar door het geven van een sneltoets voor het aanmaken van een framework met commentaar voor het documenteren van een klasse of member function. Stel, u heeft al deze code geschreven:

```
class Car {
  public:
    std::string get_color () const;
};
```

U wilt nu documentatie toevoegen aan zowel de klasse en de memberfunctie. Plaats de cursor op de eerste regel en selecteer **Code** → **Document declaratie** of toets **Alt-Shift-D**. KDevelop zal als volgt reageren:

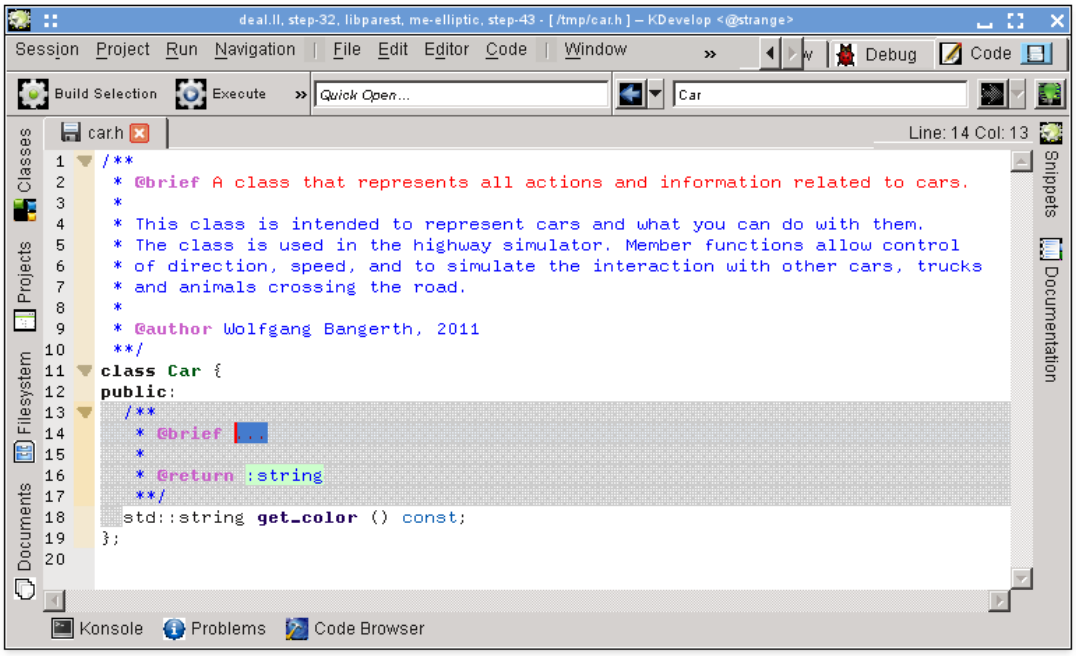


De cursor staat al in het lichtgrijze gebied waar u een korte beschrijving kunt geven (na het doxygen trefwoord `@brief`) van deze klasse. U kunt vervolgens uitgebreider documentatie toevoegen aan dit commentaar die meer gedetailleerde informatie geeft over de klasse:

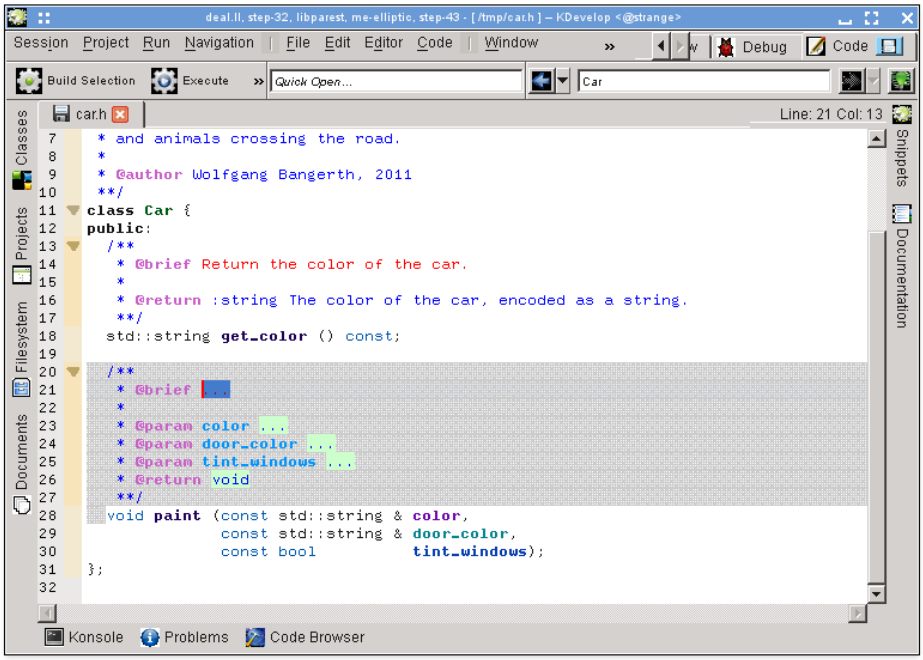


Als het commentaar is geselecteerd in de editor, dan zal de text daarvan groen oplichten (de tekst zal niet meer oplichten als de cursor van het commentaar weg is). Als u aan het eind van de regel de **Enter**-toets indrukt dan start KDevelop automatisch een nieuwe regel met een sterretje en de cursor een karakter ingesprongen.

Nu gaan we de memberfunctie documenteren, we plaatsen opnieuw de cursor op de regel met de declaratie en selecteren **Code** → **Document declaratie** of toetsen **Alt-Shift-D**:



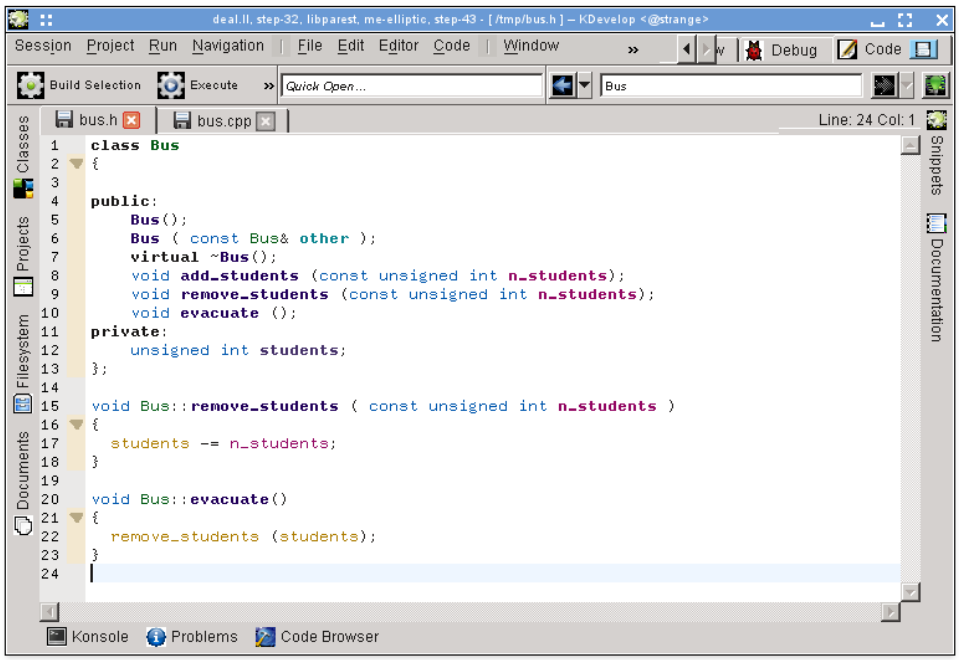
Opnieuw genereert KDevelop automatisch het skelet voor een commentaar inclusief de documentatie voor de functie zelf en het return type. In dit geval geeft de naam van de functie vrij duidelijk het doel weer, maar vaak genoeg is dat niet het geval bij de argumenten van een functie en moeten elk apart gedocumenteerd worden. Om dit duidelijker te maken bekijken we een iets interessantere functie en de documentatie die KDevelop automatisch genereert:



De voorgestelde documentatie heeft bijvoorbeeld al alle nodige Doxygen-velden voor de individuele parameters.

3.4.4 Hernoemen van variabelen, functies en klassen

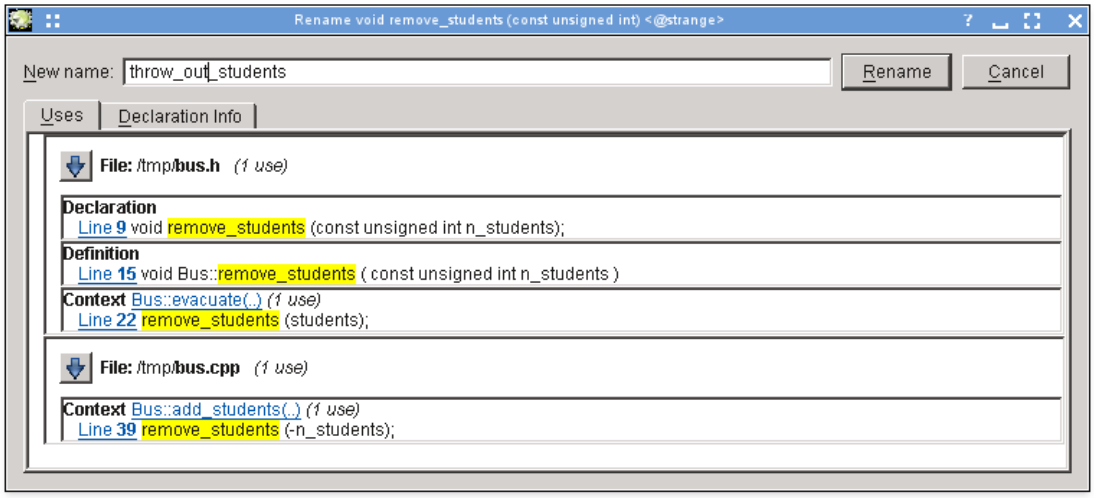
Soms wilt u een functie, klasse of een variabele hernoemen. Stel bijvoorbeeld dat we het volgende al hebben:



We realiseren vervolgens dat we ongelukkig zijn met de naam `remove_students` en dat de naam (bijvoorbeeld) `throw_out_students` beter is. We zouden een zoek-en-vervang actie voor de naam kunnen uitvoeren, maar dit heeft twee nadelen:

- De functie kan voor meer dan één bestand worden gebruikt.
- We willen alleen deze functie hernoemen en niet de functies die de dezelfde naam hebben in andere klasses of namespaces.

Beide problemen kunt u oplossen door de cursor op een locatie te plaatsen waar deze voorkomt en vervolgens **Code** → **Declaraties hernoemen** te selecteren (of met de rechtermuisknop te klikken op de naam en **Bus::remove_students hernoemen** selecteren). Er opent een dialoogvenster waarin u de nieuwe naam van de functie kunt invoeren en waar u ook alle plaatsen ziet waar deze daadwerkelijk is gebruikt:



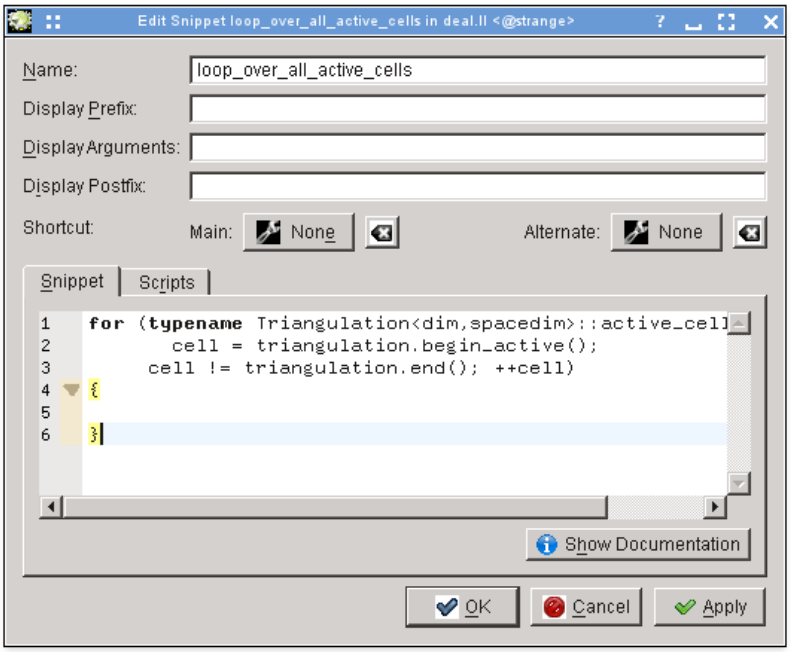
3.4.5 Codefragmenten

In de meeste projecten zijn er stukken code die vaak voorkomen in de broncode. Voorbeelden zijn: voor schrijvers van compilers, een loop met alle instructies; voor schrijvers van user interfaces, controle van de gebruikersinvoer op geldigheid en indien niet geldig het vervolgens geven van een foutmelding; in het project van de auteur van deze tekst, zou het de volgende code kunnen zijn:

```
for (typename Triangulation::active_cell_iterator
       cell = triangulation.begin_active();
     cell != triangulation.end(); ++cell)
  ... doe iets met de cel ...
```

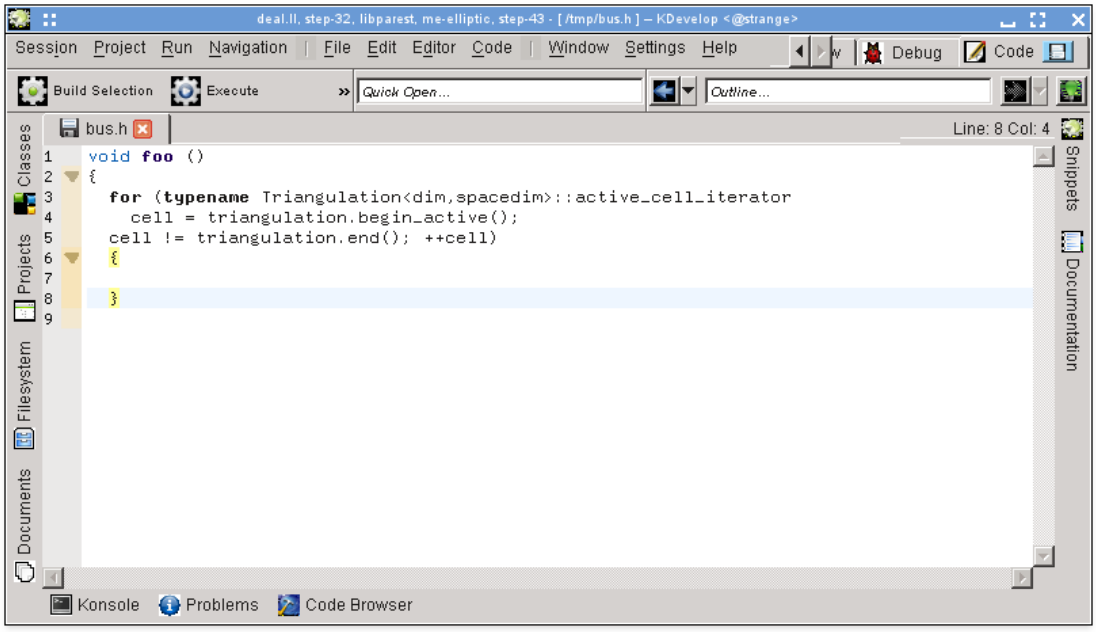
In plaats van dat u dit soort teksten iedere keer opnieuw intypt (met het risico van tikfouten), kan het hulpmiddel **Fragmenten** van KDevelop u daarbij helpen. Open het venster (lees Weergave van hulpmiddelen als u het desbetreffende knop nog niet aantreft rondom het hoofdvenster). Klik vervolgens op de knop 'Repository toevoegen' (een beetje verkeerde naam — hiermee kan u de naam bepalen voor een verzameling van fragmenten mee aanmaken voor bepaalde soorten broncodes, bijv. C++ sources) en vervolgens een lege repository aanmaken. Klik vervolgens op

om een fragment toe te voegen, zodat u het volgende dialoogvenster te zien krijgt:



> **OPMERKING**
> De naam van het fragment mag geen spaties of andere speciale karakters hebben omdat het op de naam van een normale functie of variabele moet lijken (vanwege redenen die duidelijker worden in de volgende paragraaf).

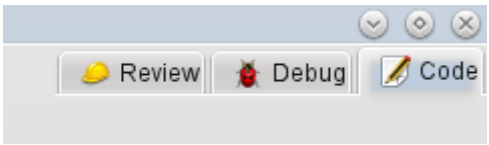
Om een gedefinieerde fragment te gebruiken tijdens het bewerken hoeft u alleen maar de naam van het fragment te typen zoals u dat met een andere functie of variabele zou doen. Deze naam komt beschikbaar voor automatische aanvulling — wat inhoud dat het geen probleem is als u een lange omschrijvende naam voor een fragment gebruikt zoals hierboven — en als u door de automatische aanvulling voorgestelde suggestie accepteert (bijvoorbeeld door op **Enter** te drukken), dan zal de al ingevoerde gedeelte van de naam van het fragment worden vervangen door de volledige tekst van het fragment met de juiste inspringingen:



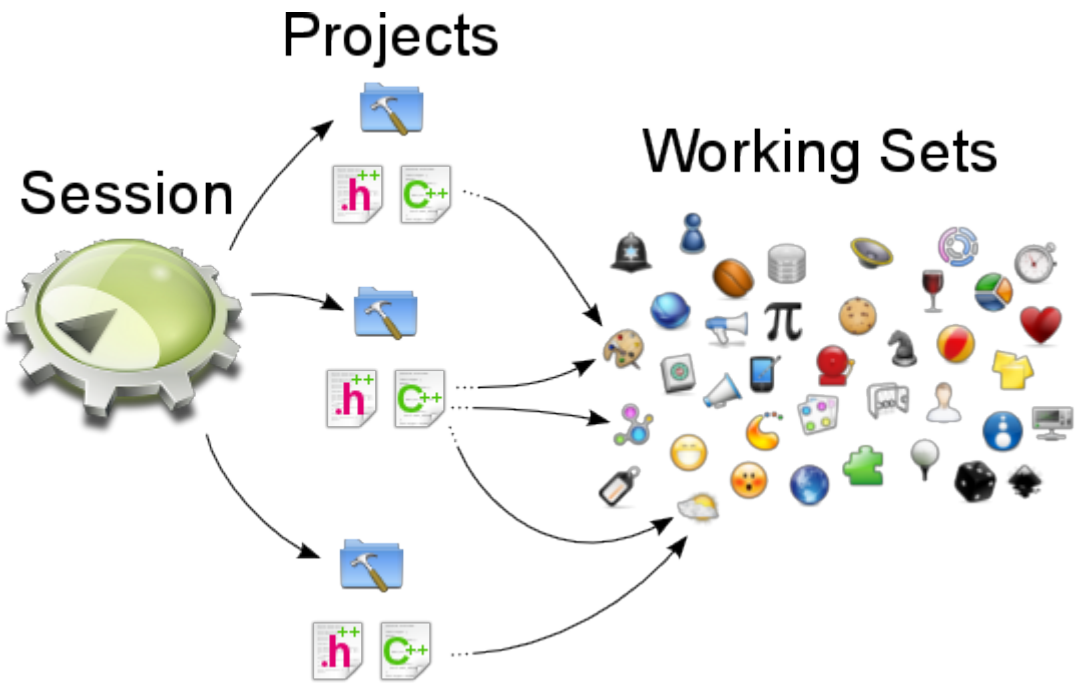
Merk op dat voor de juiste werking hiervan is het niet nodig dat het **Fragmenten**-dialoogvenster geopend of zichtbaar is: u heeft het dialoogvenster alleen nodig om nieuwe fragmenten te definiëren. Als alternatieve, maar minder handige, manier om de volledige tekst van een fragment te krijgen is door te klikken in het desbetreffende dialoogvenster.

> OPMERKING
> Met fragmenten is veel meer mogelijk dan hier uitgelegd. Lees gedetailleerde documentatie van het hulpmiddel voor fragmenten voor een complete uitleg van de mogelijkheden.

3.5 Modes en werksets

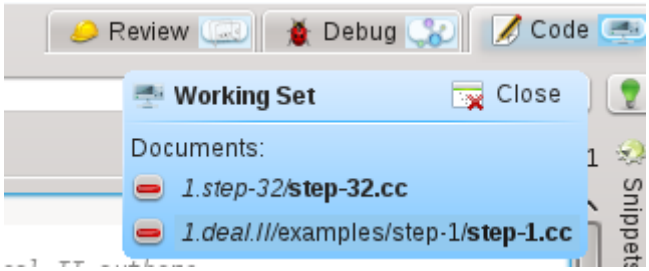


Als u tot hier bent gekomen dan is het nu tijd om in de rechterbovenkant van het hoofdvenster van KDevelop te kijken: Zoals u in de afbeelding ziet, zijn er drie **modes** KDevelop kan in de volgende modus zijn: **Code** (deze modus beschrijven we in dit hoofdstuk over het bewerken van broncode), **Opstarten van debuggen** (lees Programma's debuggen) en **Huidig project vastleggen** (lees Werken met versiebeheersystemen).



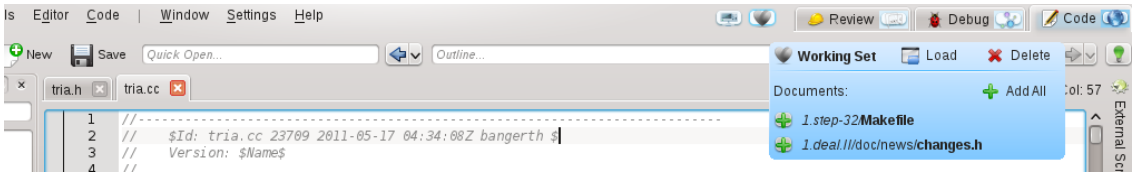
elke modus heeft zijn eigen set hulpmiddelen die u rondom het hoofdvenstervenster kunt vinden, en elke modus heeft ook een *werkset* met de huidige geopende bestanden en documenten. Alle werksets horen bij de huidige sessie, bijv. we hebben de bovengenoemde relaties. Merk op dat de bestanden in de werkset weliswaar uit dezelfde sessie komen, maar dat ze uit verschillende projecten van deze sessie kunnen komen.

Als u KDevelop voor de eerste keer opent dan is de werkset nog leeg — er zijn nog geen bestanden geopend. Maar als u begint met het openen van bestanden on deze te bewerken (of debuggen, of nakijken en vastleggen in de andere modes) dan groeit u werkset. Het feit dat uw werkset niet leeg is ziet u aan een symbool in de tab, zoals hieronder te zien . Merk op dat iedere keer als u KDevelop afsluit en later weer opstart, de werkset wordt opgeslagen en weer geopend, bijv. u krijgt dezelfde verzameling geopende bestanden.



Als uw muis boven het symbool van de werkset zweeft dan krijgt u een tooltip met de in deze werkset geopende bestanden (hier: de bestanden `step-32.cc` en `step-1.cc`). Door te klikken op het rode minus-tekentje sluit u de tab voor het desbetreffende bestand. Belangrijker is dat u door te klikken op knop genaamd **sluiten** de hele werkset sluit(bijv. u sluit alle op dit moment geopende bestanden). Het idee van het sluiten van een werkset is dat niet gewoon alle bestanden sluit, maar dat de werkset wordt opgeslagen en een nieuwe (nog lege) geopend. U ziet dat hier:

Let op de twee symbolen links van de drie mode tabs (het hart en het onherkenbare symbool links daarvan). Alle twee symbolen stellen een opgeslagen werkset voor, naast de op dit moment geopende werkset. Als u met uw muis boven het hart-symbool zweeft dan ziet u zoiets:



Hier ziet u dat de desbetreffende werkset twee bestanden bevat met de namen: `Makefile` en `changes.h`. Als u klikt op **Laden** dan sluit u de huidige werkset en slaat deze op (waarin zoals u hier ziet de bestanden `tria.h` en `tria.cc` geopend zijn) en opent in plaatst daarvan de geselecteerde werkset. U kunt ook een werkset permanent verwijderen,waardoor het ook verdwijnt uit de set met opgeslagen werksets.

3.6 Enige nuttige sneltoetsen

De editor van KDevelop heeft de standaard sneltoetsen voor alle gebruikelijke bewerkingen. Daarnaast heeft het een aantal voor meer ingewikkelder bewerkingen van broncode, sommige daarvan hebben hun eigen sneltoetsen. Hieronder volgt de uitleg van enkele veel gebruikte:

Rond code springen	
Ctrl-Alt-O	Snel openen van bestand: voer een gedeelte van een bestandnaam in en selecteer het gewenste bestand tussen alle bestanden in de projectmappen van de huidige sessie die overeenkomen met de ingevoerde tekenreeks; dit bestand zal vervolgens worden geopend.
Ctrl-Alt-C	Snel openen van klasse: voer een gedeelte van een klassenaam in en selecteer de gewenste klasse tussen alle klassen die overeenkomen met de ingevoerde tekenreeks; de cursor zal vervolgens naar de klassedeclaratie springen.
Ctrl-Alt-M	Snel openen van functie: voer een gedeelte van een functienaam in en selecteer de gewenste functie tussen alle functies die overeenkomen met de ingevoerde tekenreeks; de cursor zal vervolgens naar het geselecteerde item springen. Merk op dat in de lijst zowel de declaraties als de definities te zien zijn.
Ctrl-Alt-Q	Snel openen: type iets (bestandsnaam, klassenaam, functienaam) en krijg een lijst met dingen die daarmee overeenkomen.
Ctrl-Alt-N	Uitlijning: geeft een lijst met alle zaken die in dit bestand voorkomen, bijv. klassedeclaraties en functiedefinities

Ctrl-,	Ga naar de definitie van een functie als de cursor op dit moment op een functiedeclaratie staat.
Ctrl-.	Ga naar de declaratie van een functie als de cursor op dit moment op een functiedefinitie staat.
Ctrl-Alt-PageDown	Naar volgende functie springen
Ctrl-Alt-PageUp	Naar vorige functie springen
Ctrl-G	Ga naar regel

Zoeken en vervangen	
Ctrl-F	Zoeken
F3	Volgende zoeken
Ctrl-R	Vervangen
Ctrl-Alt-F	Zoek en vervang in meerdere bestanden

Andere zaken	
Ctrl-_	Een niveau inklappen: verwijder dit blok uit het zicht, bijvoorbeeld als u zich wilt concentreren op het grotere geheel wilt van een functie wilt concentreren.
Ctrl-+	Een niveau uitklappen: het inklappen ongedaan maken
Ctrl-D	Commentaar maken van geselecteerde tekst of regel
Ctrl-Shift-D	Geselecteerde tekst of regel als commentaar verwijderen
Alt-Shift-D	Documenteer de geselecteerde functie. Als de cursor op een functie- of klassedeclaratie staat dan wordt een commentaar in doxygen-style aangemaakt gevuld met een lijst van alle parameters return values, etc.
Ctrl-T	Het huidige en de vorige teken omwisselen
Ctrl-K	Verwijdert de geselecteerde regel (letop: dit is niet de ‘delete from here to the end of the line’ van emacs)

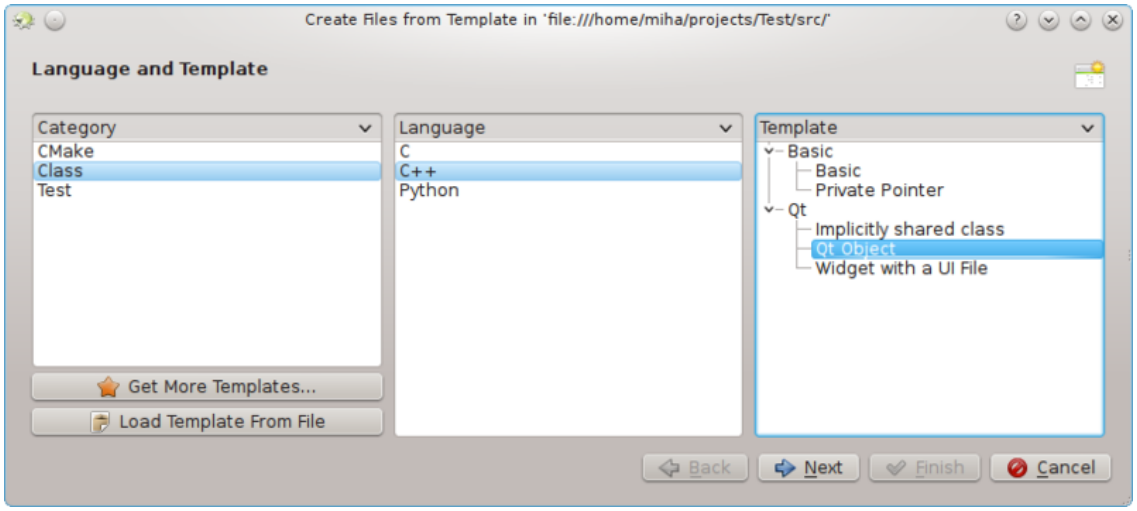
Hoofdstuk 4

Code aanmaken met templates

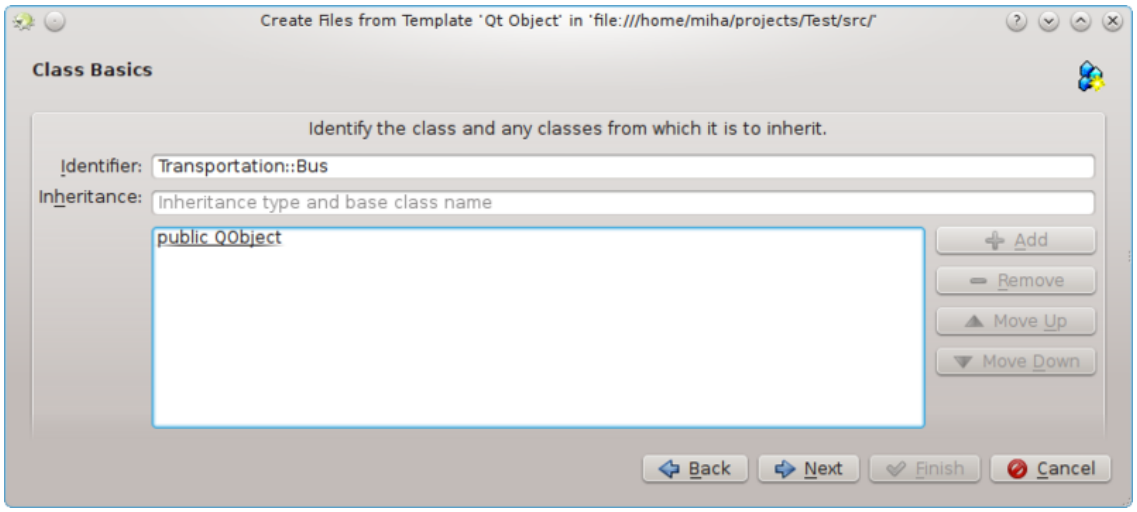
KDevelop gebruikt templates voor de aanmaak van bronbestanden en om het herhalende schrijven van code te vermijden.

4.1 Een nieuwe klasse aanmaken

Het genereren van code wordt het meest gebruikt bij het schrijven van nieuwe klassen. Voor de creatie van een nieuwe klasse in een bestaand project klikt u met de rechtermuisknop op de projectmap en selecteert **Vanuit een sjabloon maken...**. Dit dialoogvenster kunt u ook openen via het menu door **Bestand** → **Nieuw van sjabloon** te selecteren, maar het gebruik van een projectmap heeft het voordeel dat het een locatie instelt voor de uitvoerbestanden. Selecteer `Klasse` in het keuzemenu, en de gewenste programmeertaal en sjabloon in het volgende dialoogvenster. Na de selectie van een sjabloon voor een klasse, vult u de details van de nieuwe klasse in.

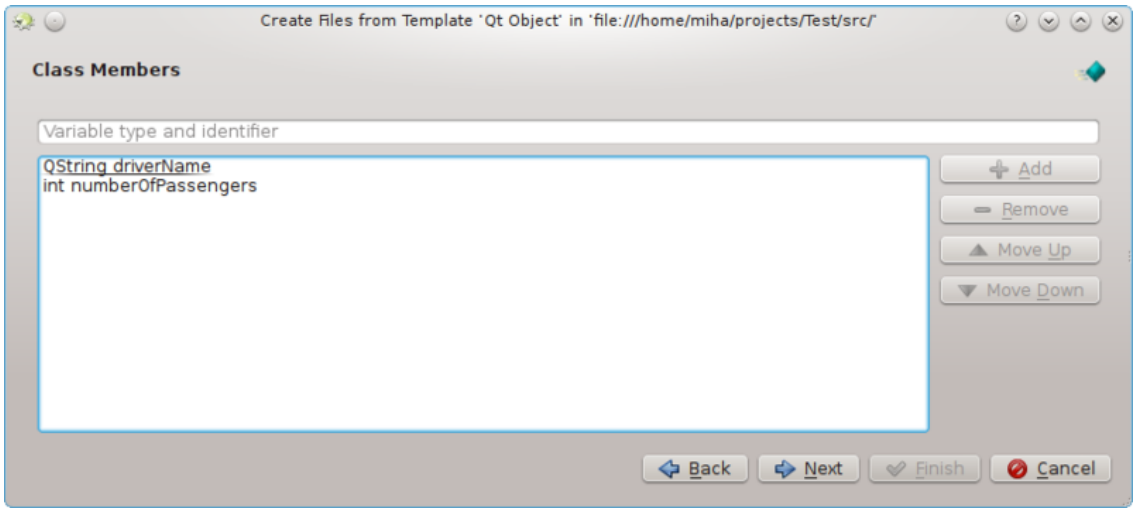


Eerst moet u een identificatie opgeven voor de nieuwe klasse. Dit kan een simpele naam zijn (zoals `Bus`) of een complete identificatie met namespaces (zoals `Transportation::Bus`). In het laatste geval zal KDevelop de identifier analyseren en de namespaces van de eigenlijke naam separeren. Op dezelfde pagina kunt u basisklasses voor de nieuwe klasse toevoegen. U zal merken dat met sommige sjablonen gelijk een basisklasse wordt toegevoegd, niets houd u tegen om deze weer te verwijderen en/of andere klassen toe te voegen. U moet de volledige overervings-statement hier uitschrijven, deze is programmeertaal afhankelijk, zoals als `public QObject` for C++, `extends SomeClass` voor PHP of eenvoudig de klassenaam voor Python.



In het volgende venster kunt u kiezen uit een lijst met virtuele methoden van all overgeërfde klassen, met enkele standaard constructors, destructors en operators. Het selecteren van een keuzevakje naast de omschrijving zal deze methode in de nieuwe klasse implementeren.

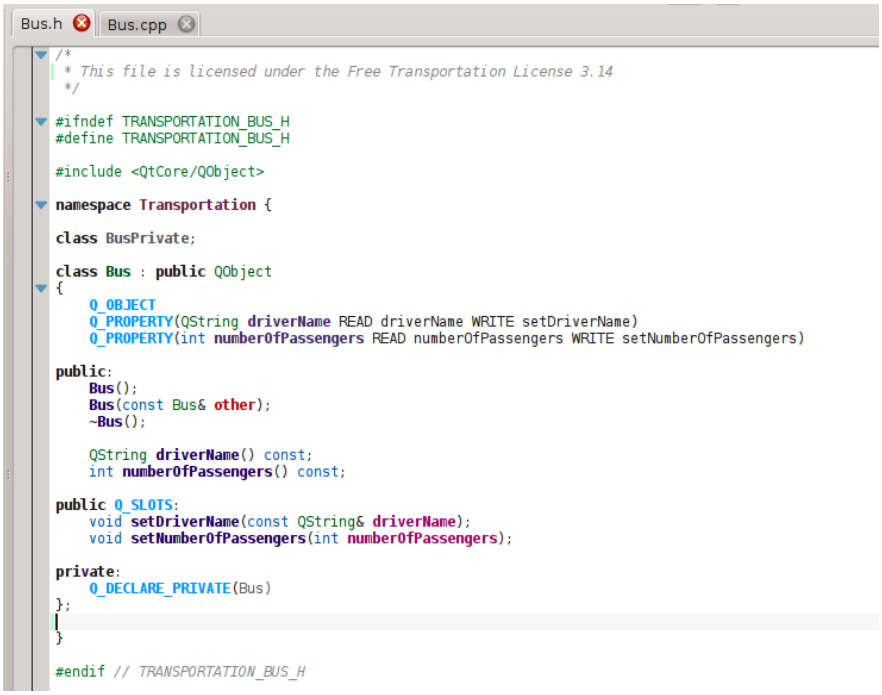
Als u klikt op **Volgende** dan opent een dialoogvenster waar u members aan de klas kunt toevoegen. Afhankelijk van het geselecteerde sjabloon kunnen deze in de nieuwe klasse verschijnen als nieuwe member variabelen of als eigenschappen met setters en getters daarvoor. In een programmeertaal waar variabelen gedeclareerd moeten worden zoals C++, moet u zowel de naam en het type van de member specificeren, zoals `int number` of `QString name`. In andere programmeertalen kunt u het type weglaten, maar is het gebruikelijk om dit toch wel te doen omdat het gekozen sjabloon er gebruik van kan maken.



In het volgende dialoogvenster kunt u een licentie voor uw nieuwe klasse invoeren, door de gekozen sjabloon vereiste speciale opties en de uitvoerlocaties voor de aangemaakte bestanden instellen. Door te klikken op **Voltooien**, sluit u de assistent en creëert u de nieuw klasse. De gecreëerde bestanden worden geopend in de editor en u kunt onmiddellijk beginnen met het toevoegen van code.

Na het creëren van de nieuwe C++ klasse heeft u de keuzemogelijkheid om het toe te voegen aan een project. Kies een project uit in het dialoogvenster of sluit het venster en voeg een doel later handmatig toe.

Als u heeft gekozen voor het `Qt Object`-sjabloon en enkele van de standaard methoden heeft gekozen en twee member variabelen heeft toegevoegd dan moet het resultaat eruit zien als in de volgende afbeelding.



U kunt zien dat de data members zijn omgezet naar Qt properties met accessor functies en de Q_PROPERTY macros. Argumenten naar setter functies worden zelf waar mogelijk doorgegeven met const-reference. Bovendien is een private klasse gedeclareerd, een private pointer gecreëerd met Q_DECLARE_PRIVATE. Dit is allemaal uitgevoerd door het sjabloon, in de eerste stap een andere sjabloon uitkiezen zou het resultaat compleet anders zijn.

4.2 Een nieuwe unit test creëren

Ofschoon de meeste test frameworks vereisen dat elke test ook een klasse is, heeft KDevelop een methode die het makkelijker maakt om unit tests te creëren. Om een nieuwe te creëren klikt u met de rechtermuisknop op een projectmap en selecteert **Vanuit een sjabloon maken...**. In het sjabloon-selectiepagina selecteert u de categorie `Test`, vervolgens kiest u de programmeertaal en sjabloon en klik op **Verder**.

U wordt gevraagd om een naam voor de test en een lijst met test cases. Voor de test cases hoeft u alleen maar een lijst met namen op te geven. Sommige unit testing frameworks zoals PyUnit en PHPUnit, vereisen dat test cases starten met een speciale voorvoegsel. In KDevelop, is het sjabloon verantwoordelijk voor te toevoegen van het voorvoegsel, daarom hoeft u zelf het voorvoegsel toe te voegen. Na het klikken op **Volgende**, kunt een licentie en de uitvoerlocaties voor de aangemaakte bestanden invoeren waarna KDevelop de test aanmaakt.

Op deze manier aangemaakte unit tests worden niet automatisch toegevoegd aan een target. Als u CTest of een andere testing framework gebruikt dan moet u zelf de nieuwe bestanden aan de target toevoegen.

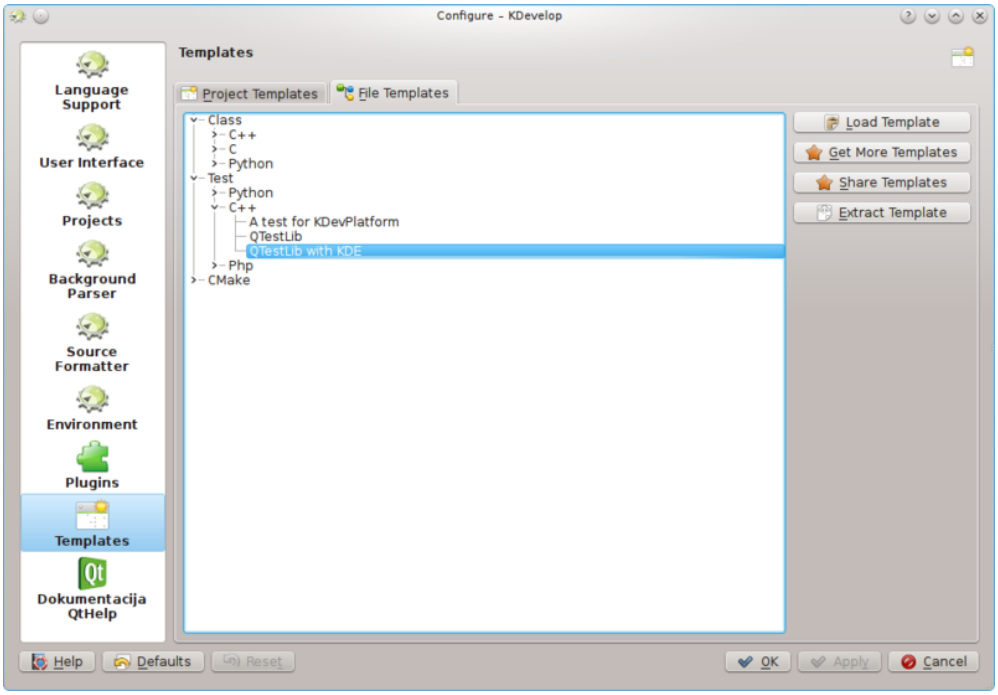
4.3 Andere bestanden

Terwijl klassen en unit tests extra aandacht krijgen bij het genereren van code via sjablonen, kan u deze methode ook gebruiken bij andere vormen van bronbestanden. U kunt bijvoorbeeld ook een sjabloon gebruiken voor een CMake Find module of een .desktop-bestand. Dit kunt u doen door bij **Vanuit een sjabloon maken...** de gewenste categorie en sjabloon te selecteren. Als de geselecteerde categorie niet `Klasse` en niet `Test`is dan hoeft u alleen de licentie, de bij het sjabloon

horende speciale opties en de locaties voor de uitvoerbestanden in te voeren. Net zoals bij de klasses en tests zullen na het sluiten van de assistent de nieuwe bestanden worden gecreëerd en vervolgens geopend in de editor.

4.4 Sjablonen beheren

In het **Bestand** → **Nieuw van sjabloon...** assistent kunt u ook extra sjabloonbestanden downloaden door te klikken op de knop **Meer sjablonen ophalen...**. Er opent een Get Hot New Stuff dialoogvenster waarmee u extra sjablonen kunt installeren, bijwerken of verwijderen. Er is voor sjablonen ook een configuratiemodule, u kunt deze bereiken via **Instellingen** → **KDevelop instellen** → **Sjablonen**. van hieruit kunt u zowel bestandsjablonen (hierboven uitgelegd) als projectsjablonen (gebruikt voor het creëren van nieuw projecten) beheren.



Als geen van de beschikbare sjablonen geschikt is voor uw project dan kunt u natuurlijk zelf nieuwe sjablonen creëren. De makkelijkste manier is waarschijnlijk een bestaande sjabloon kopiëren en deze aanpassen, er is een korte instructie en een langer document met specificaties om u daarbij te helpen. Om een geïnstalleerde sjabloon te kopiëren, opent u de sjabloon-beheerder door op **Instellingen** → **KDevelop instellen...** → **Sjablonen** te klikken, selecteer het sjabloon dat u wilt kopiëren, klik vervolgens op de knop **Sjabloon klonen**. Selecteer een map als bestemming en klik op **OK**, en het sjabloon zal naar de gekozen map worden gekopieerd. Nadat u het heeft geopend kunt u het nieuwe sjabloon gaan bewerken. Als u klaar bent met uw wijzigingen, kunt u uw nieuwe sjabloon weer in KDevelop importeren door sjabloonbeheer te openen en de van toepassing zijnde tab te openen (naar keuze **Projectsjablonen** of **Bestandsjablonen**) en klik op **Sjabloon uit bestand laden**. Open het sjabloonbeschrijvingsbestand, die een van de volgende twee bestandsextensies heeft: `.kdevtemplate` of `.desktop`. KDevelop zal de bestanden comprimeren in een sjabloon-archief en vervolgens het sjabloon importeren.

> **OPMERKING**
> Als u een bestaand sjabloon kopieert dan moet u er voor zorgen dat u het hernoemt voordat u het weer terug importeert omdat u anders het bestaande sjabloon overschrijft of eindigt met twee gelijk genaamde sjablonen. Om een sjabloon te hernoemen moet u het beschrijvingsbestand een unieke naam geven (maar blijf wel de extensie gebruiken), en het item `Naam` in het beschrijvingsbestand wijzigen.

Als u een sjabloon vanaf nul wilt schrijven dan kunt u starten met een voorbeeld C++ class sjabloon door een nieuw project creëren en het project `C++ Class Template` in de categorie `KDevelop` te selecteren.

Hoofdstuk 5

Projecten bouwen (compileren) met aangepaste Makefiles

Veel projecten beschrijven hoe bronbestanden gecompileerd moeten worden (en welke bestanden opnieuw gecompileerd moeten worden nadat een bronbestand of een headerbestand is gewijzigd) door gebruik van Makefiles die worden interpreteert door het programma **make** (lees bijvoorbeeld GNU make). Voor simpele projecten is het meestal erg makkelijk om een dergelijk bestand handmatig op te zetten. Grotere projecten hebben vaak hun Makefiles geïntegreerd met de **GNU autotools** (autoconf, autoheader, automake). In deze sectie nemen we aan dat u een Makefile voor uw project heeft en u wilt KDevelop instrueren hoe ermee om te gaan.

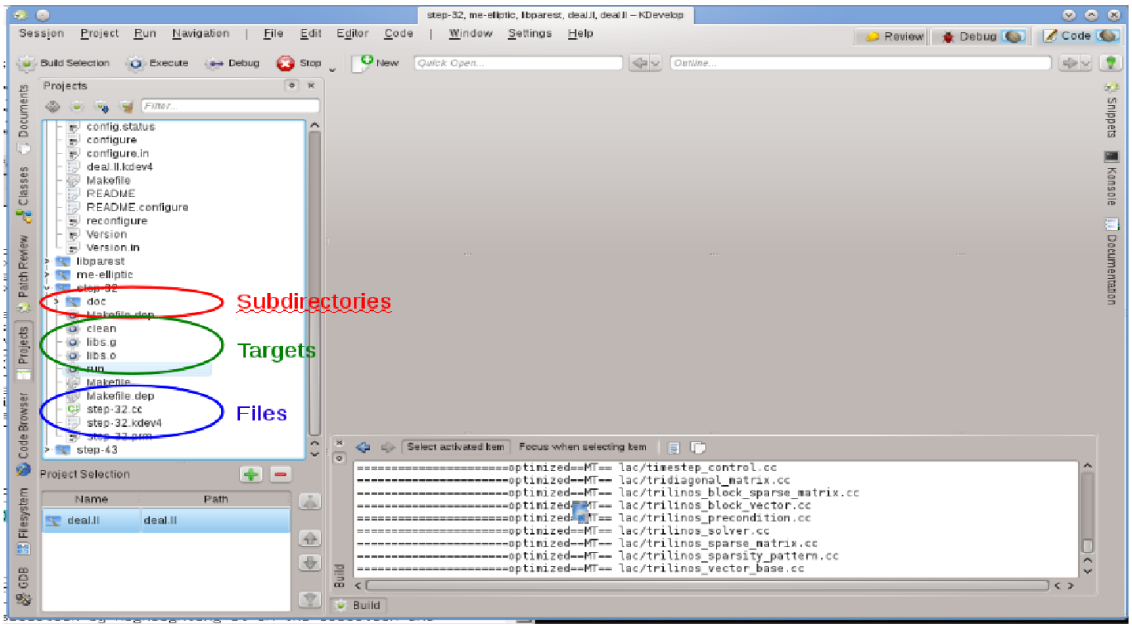
> OPMERKING
> KDevelop 4.x heeft geen kennis over **GNU autotools** op het moment dat deze sectie is geschreven. Als uw project deze gebruikt dan moet u op de commandoregel de volgende opdracht gebruiken `./configure` of een van de andere van toepassing zijnde opdrachten. Als u dit vanuit KDevelop wilt doen dan moet u het **Konsole** openen (voeg het indien nodig toe aan de vensterrand via het menu **Venster** → **Weergave van hulpmiddelen**) van waaruit u de opdracht `./configure` op de commandoregel kunt geven.

De eerste stap is het instellen van KDevelop voor de targets in uw Makefiles. Er zijn twee manieren om dat te doen: individuele Makefile targets selecteren, of een set targets selecteren die u vaak wilt compileren. bij beide benaderingen opent u het venster **Projecten** door te klikken op de knop **Projecten** langs de rand van het hoofdvenster van KDevelop (als de knop niet aanwezig is, lees dan hierboven hoe u de knop kunt toevoegen). Het venster **Projecten** heeft twee gedeelten: de bovenste helft — getiteld **Projecten** — geeft een lijst met al uw projecten waarvan u de onderliggende mappen kunt uitvouwen. De onderste helft — getiteld **Sequentie bouwen** — geeft een lijst met gedeeltes van projecten die gebouwd worden als u het menu item **Project** → **Selectie bouwen** selecteert of **F8**; toetst; we zullen hierop later terug komen.

5.1 Individuele Makefile targets bouwen

Klap in de bovenste helft van het projectvenster bij een project de mapstructuur uit, bijvoorbeeld van een project waarvan u een Makefile target wilt bouwen. Hierdoor krijgt u pictogrammen te zien voor (i) mappen in het project, (ii) bestanden op het hoogste niveau van dit project, (iii) Makefile targets dieKDevelop kan vinden. Deze categorieën ziet u rechts in het venster. Merk op dat KDevelop in bepaalde mate de Makefile syntax *begrijpt* en kan daarom targets laten zien

die in deze Makefile zijn gedefinieerd(maar dit begrip schiet tekort als de targets samengesteld of indirect zijn).



Om een van de hier opgenoemde targets te bouwen, selecteert u het eerst met de rechtermuisknop om vervolgens **Bouwen** te selecteren. Als u dit bijvoorbeeld doet met het target 'clean' dan wordt gewoon 'make clean' uitgevoerd. U kunt dit volgen in het subvenster **Bouwen** dat zich opent, u kunt dan de commando's en de resultaten daarvan zien. (dit venster hoort bij het hulpmiddel **Bouwen**, u kunt daarom dit venster sluiten en later weer heropenen met de knop **bouwen** linksonder het hoofdvenster. U kunt het aan de onderkant van de afbeelding zien).

5.2 Een verzameling van Makefile targets maken voor herhaald bouwen

Met de rechtermuisknop op individuele Makefile targets klikken iedere keer dat u iets snel wil bouwen zal snel vervelen. In plaats daarvan willen we individuele targets hebben voor een of meerdere projecten in de sessie dat we herhaaldelijk kunnen bouwen zonder veel muiswerk. Hier komt het begrip 'Bouw target set' om de hoek: dit is een verzameling van Makefile targets die iedere keer dat u de knop **Selectie bouwen** in het menu indrukt na elkaar worden gebouwd, selecteer het menu-item **Project** → **Selectie bouwen**, of druk op de sneltoets **F8**.

De lijst met geselecteerde Makefile targets kunt u vinden in de onderste helft van het venster **Projecten**.

Standaard zijn alle projecten in de selectie aanwezig maar u kunt dat naar wens aanpassen. Als voorbeeld in uw projectenlijst drie projecten aanwezig zijn (een basis bibliotheek L en de twee toepassingen A en B), maar u werkt op dit moment alleen aan project A, dan wilt u misschien project B uit de selectie verwijderen door het te markeren en de knop in te drukken. Bovendien wilt u waarschijnlijk ook dat eerst bibliotheek L en vervolgens project A wordt gebouwd door de volgorde van de items in de selectie omhoog en omlaag te verplaatsen met behulp van de knoppen aan de rechterkant van de lijst. U wilt wellicht ook een bepaalde particular Makefile target aan de de selectie toevoegen, dit kunt u doen door er met de rechtermuisknop te selecteren en vervolgens **Aan bouwset toevoegen** te selecteren, of markeer het en druk op de knop net boven de lijst met geselecteerde targets.

In KDevelop kunt u voor 'make' de gebeurtenissen instellen. Open het menu item **Project** → **Configuratie openen**. Hier kunt u bijvoorbeeld het aantal parallelle taken dat 'make' moet uitvoeren instellen — als uw computer bijvoorbeeld 8 processors heeft dan is het invoeren van 8 in dit veld een goede keus. In dit dialoog is het **Standaard make doel** een Makefile target in gebruik bij *alle* targets in de selectie.

5.3 Wat te doen met foutmeldingen

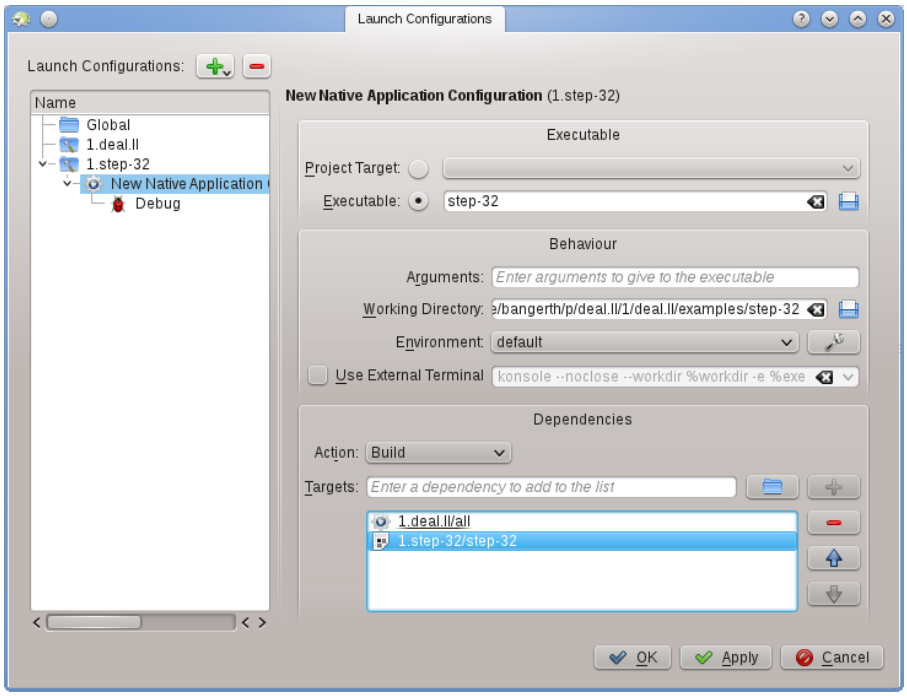
Als de compiler een foutmelding geeft, klik dan op de regel met de foutmelding en de editor zal naar de regel (en zo mogelijk naar de kolom) gaan waar de fout is ontstaan. Afhankelijk van de foutmelding zal KDevelop proberen een voorstel te geven voor reparatie van de fout, bijvoorbeeld het declareren van een nog niet gedeclareerde variabele als een onbekend symbool is gevonden.

Hoofdstuk 6

programma's uitvoeren in KDevelop

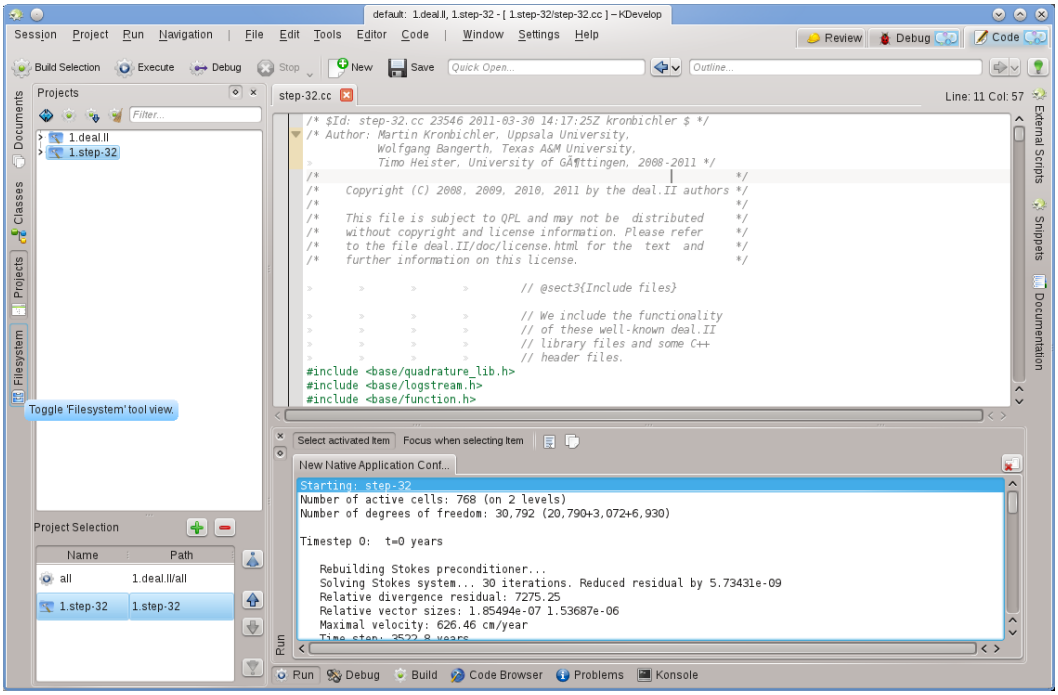
Als u een programma heeft gebouwd dan wilt u het natuurlijk opstarten. Om dit te kunnen doen, moet u eerst *Programmastarters* voor uw projecten instellen. Een *Starter* bestaat uit de naam van een uitvoerbaar bestand, een set command line parameters en een execution environment (zoals 'voer dit programma in een shell uit', of 'voer dit programma in de debugger uit').

6.1 Een programmastarter instellen in KDevelop



Om dit in te stellen gaat u naar **Uitvoeren** → **Starters instellen**, markeer het project waar u een starter voor wilt toevoegen en klikt op de knop . Nu kunt u de naam van het programma opgeven en het pad van waar u het programma wilt uitvoeren. als eerst het programma en/of andere bibliotheken gecompileerd moeten worden voor het starten van het programma dan kunt u dit in een lijst onderaan opgeven: selecteer in het keuzemenu **Bouwen**, en druk op het symbool

 rechts van het tekstvak en selecteer het doel dat u gebouwd wilt hebben. In voorbeeld hierboven heb ik het target **all** uit hetproject `1.deal.II` en `step-32` uit het project `1.step-32` geselecteerd om zeker te zijn dat de laatste versie van zowel de basis-bibliotheek en het programma zelf gecompileerd zijn voor het opstarten van het programma. Hier kunt u ook opgeven dat u het programma in de debug-mode wilt opstarten door op het **Debug** symbool te klikken en vervolgens het debug-programma op te geven; deze stap is niet nodig als dit het standaard debug-programma is dat bij het systeem hoort (bijv. gdb bij Linux®).



U kunt nu proberen of u het programma kunt uitvoeren: Selecteer **Uitvoeren** → **Starter uitvoeren** uit het hoofdmenu van KDevelop (of toets **Shift-F9**) en uw programma zou in een apart subvenster van KDevelop moeten starten . De afbeelding hierboven toont het resultaat: het nieuwe venster **Uitvoeren** onderaan toont de uitvoer van het programma dat nu is gestart, in dit geval het programma `step-32`.

> OPMERKING
> Als u meerdere startopdrachten hebt ingesteld kunt u kiezen welke uitgevoerd zou moeten worden wanneer u op **Shift-F9** drukt door te gaan naar **Uitvoeren** → **Huidige instellingen voor starten**. Er is geen gemakkelijke manier om de naam van een instelling te bewerken, echter: in het dialoogvenster dat u krijgt wanneer u **Uitvoeren** → **Huidige instellingen voor starten** kiest, dubbelklik op de naam van de configuratie in de boomstructuur links, die u in staat zal stellen om de naam van de configuratie te bewerken.

6.2 Enige nuttige sneltoetsen

Een programma uitvoeren	
F8	Compileren (roept make aan)
Shift-F9	Uitvoeren

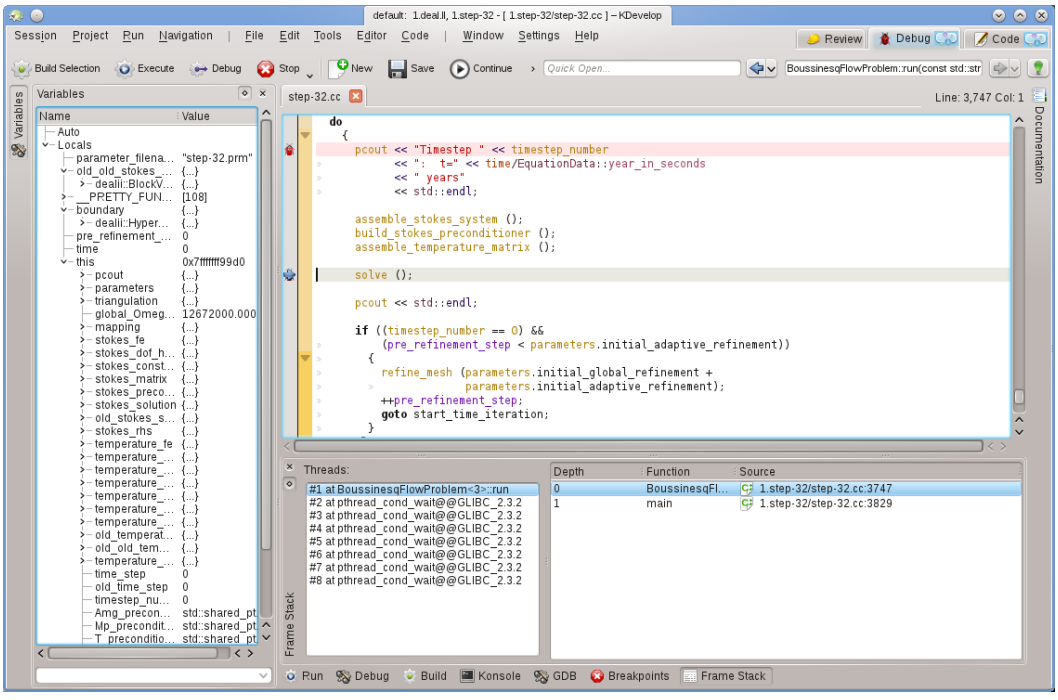
Alt-F9	Voert het programma uit in de debugger: u wilt misschien eerst breakpoints instellen, bijvoorbeeld door met de rechtermuisknop op een bepaalde regel in de broncode te klikken.

Hoofdstuk 7

Programma's debuggen in KDevelop

7.1 Een programma uitvoeren in de debugger

Nadat u een starter heeft ingesteld (lees programma's uitvoeren), kunt u het ook uitvoeren in een debugger: Selecteer het menu item **Uitvoeren** → **Opstarten debuggen** , of toets **Alt-F9**. als u bekend bent met gdb, het effect is hetzelfde als het starten van gdb met het programma opgegeven in de opstartconfiguratie en vervolgens de opdracht `Run` geven. Dit houdt in dat als het programma ergens `abort()` aanroept (bijv. wanneer u een ongeldige statement tegenkomt) of als er een segmentation fault is, dan zal de debugger stoppen. Maar als het programma gewoon tot aan zijn eind doorloopt (met of zonder het gewenste resultaat) dan zal de debugger niet uit zichzelf stoppen voordat het programma zelfstandig is geëindigd. In dit laatste geval wilt u waarschijnlijk voordat u de debugger start breakpoints zetten op alle programma-regels waarvan u wilt dat de debugger er stopt. U kunt dit doen door met de cursor naar de betreffende regel te gaan en vervolgens het menu item **Uitvoeren** → **Breekpunt aan/uit** te selecteren, of met de rechtermuisknop op een regel te klikken en vervolgens **Breekpunt aan/uit** uit het contextmenu te selecteren.

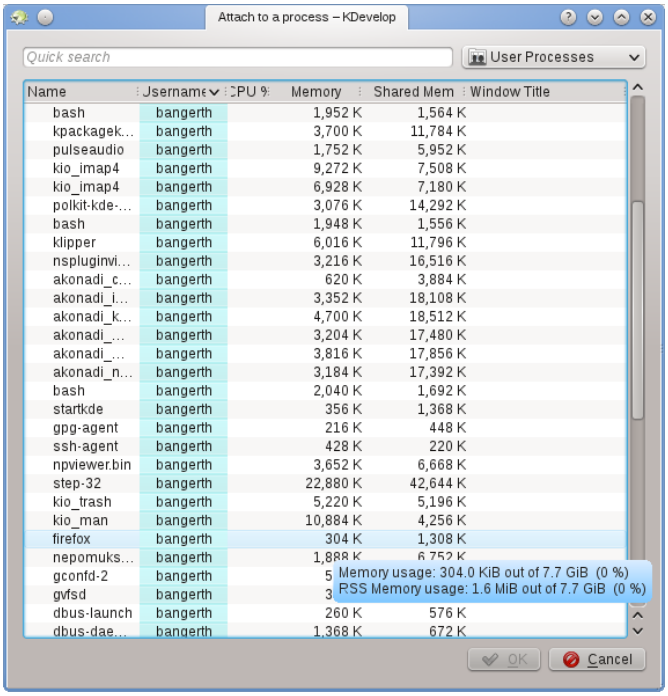


Een programma uitvoeren in de debugger zet KDevelop in een andere modus: het zal alle knoppen voor 'Hulpmiddelen' aan de randen van het hoofdvenster vervangen door diegenen die van belang zijn voor debuggen, in plaats van voor bewerken. U kunt zien in welke modus u bezig bent door naar rechtsboven van het venster te kijken: er zijn tabbladen genaamd **Nakijken**, **Debug** en **Code**; er op klikken stelt u in staat tussen de drie modi te schakelen; elke modus heeft een eigen set hulpmiddelen, die u op dezelfde manier kunt instellen zoals we de hulpmiddelen voor **Code** in de sectie Gereedschappen en vensters.

Nadat de debugger stopt (op een breekpunt of een punt waar `abort()` wordt aangeroepen) kunt u een variëteit van informatie inspecteren over uw programma. In de bovenstaande afbeelding, bijvoorbeeld, hebben we onderaan het hulpmiddel **Framestack** geselecteerd (ruwweg gelijk aan 'backtrace' van gdb en commando's 'info threads') die de verschillende threads die nu actief zijn in uw programma aan de linkerkant (hier een totaal van 8) en hoe uitvoeren leidde naar het huidige stoppunt rechts (hier: `main()` genaamd `run()`; de lijst zou langer zijn als we gestopt waren in een functie genaamd door `run()` zelf). Links kunnen we lokale variabelen inspecteren inclusief het huidige object (het object aangewezen door de variabele `this`).

Vanaf hier zijn er verschillende mogelijkheden waaruit u kunt kiezen: U kunt de geselecteerde regel uitvoeren (**F10**, gdb's 'next' commando), de functie inspecteren (**F11**, gdb's 'step' commando), of ga naar het einde van de functie (**F12**, gdb's 'finish' commando). KDevelop update voortdurend de variabelen aan de linkerkant bij met de actuele waarden. Als u met uw muis boven een symbool (bijv. een variabele;) in uw code zweeft , dan toont KDevelop u de actuele waarde daarvan en krijgt u de mogelijkheid aangeboden om het programma te stoppen als de waarde daarvan wijzigt. Als u bekent met gdb dan kunt ook op de knop **GDB** klikken zodat u daarna een opdracht voor gdb kunt invoeren, bijvoorbeeld om de waarde van een variabele te wijzigen (hiervoor lijkt geen andere methode te bestaan).

7.2 De debugger aan een lopend programma vastkoppelen



Soms wilt u een programma dat al is opgestart debuggen. Een toepassing hiervan is het debuggen van parallelle programma's door middel van MPI, of het het debuggen van een lang

lopend achtergrond proces. Hiervoor gaat u naar het menu-item **Uitvoeren** → **Aan proces hechten**, waarna zoals hierboven te zien een venster opent. U kunt hier het programma selecteren dat overeenkomt met het project dat nu in KDevelop geopend is - in mijn geval zou dat het programma step-32 zijn.

Deze lijst met programma's kan u in verwarring brengen omdat het vaak net zoals in bovenstaand voorbeeld erg lang is. U kunt uzelf het een beetje makkelijker maken door naar het keuzemenu rechtsboven van het venster te gaan. De standaard waarde is **Gebruikersprocessen**, bijv. alle programma's die door een van de ingelogde gebruikers wordt gebruikt (als dit uw desktop of laptop is dan bent u waarschijnlijk de enige gebruiker, afgezien van root en verschillende service accounts); de lijst toont de processen van root niet, maar u kunt de lijst inkorten door de keuze **Eigen processen**, hierdoor zijn alle programma's van andere gebruikers niet meer zichtbaar. Of nog beter: selecteer **Alleen programma's**, hierdoor zijn een heleboel processen die formeel onder uw naam draaien maar waar u meestal geen contact mee heeft niet meer zichtbaar, zoals de window manager, taken op de achtergrond en andere die onwaarschijnlijke kandidaten voor debugging zijn.

Als u een proces heeft geselecteerd en daaraan vastgehecht dan zal KDevelop's overschakelen naar de debug-modus, alle gebruikelijke debugger-vensters openen en het programma onderbreken op het moment dat u eraan vast koppelde. Hierna wilt u waarschijnlijk breakpoints, viewpoints, of wat er verder noodzakelijk is instellen om daarna via **Uitvoeren** → **Doorgaan** het programma verder te laten gaan.

7.3 Enige nuttige sneltoetsen

Debugging	
F10	Overslaan van (gdb's 'next')
F11	Ga naar (gdb's 'step')
F12	Beëindig (gdb's 'finish')

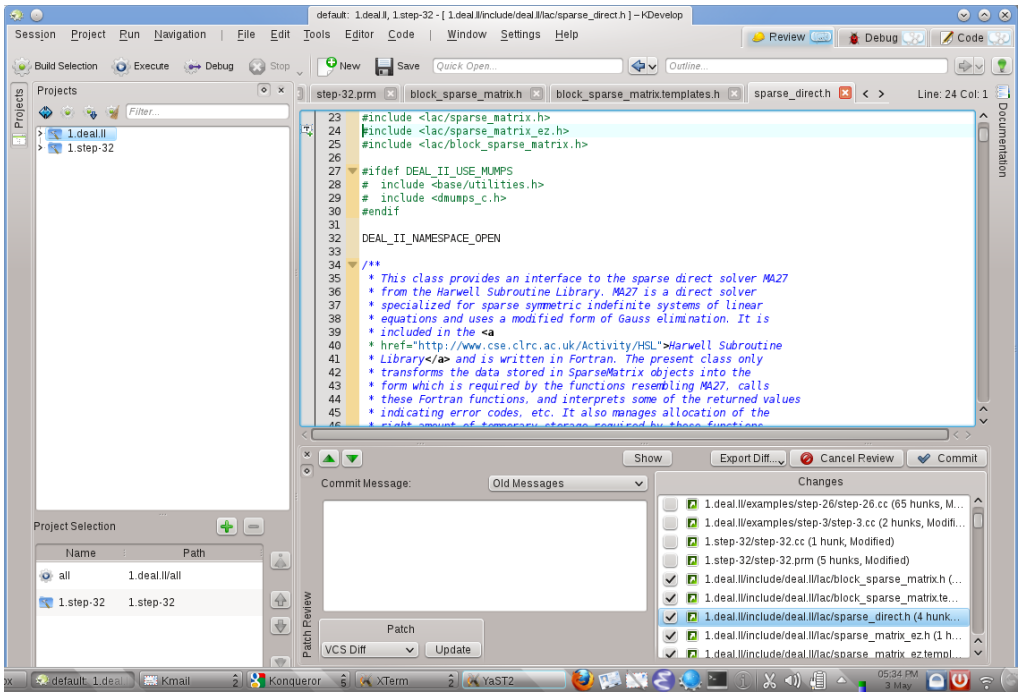
Hoofdstuk 8

Werken met versiebeheersystemen

als u met grotere projecten werkt dan is de kans groot dat de broncode beheert wordt via een versiebeheersysteem zoals subversion of git. De volgende beschrijving is geschreven met **subversion** in gedachte maar is ook van toepassing als u **git** of een ander ondersteund versiebeheersysteem gebruikt.

Vergeet niet dat als de map waarin een project is opgeslagen onder beheer is via een versiebeheersysteem dat KDevelop dit automatisch detecteert. Met andere woorden: het is dus niet nodig om opdracht te geven aan KDevelop om een kopie uit te checken wanneer u een project opzet; u hoeft KDevelop alleen maar naar de map te laten kijken waarnaar u eerder een kopie uitgecheckt. Als u een dergelijke map onder controle van versiebeheersysteem heeft, open de werkbalk **Projecten** links. U kunt hier een aantal dingen die u kunt doen:

- Als uw map verouderd is dan kunt het bijwerken vanuit de repository: klik op de naam van het project met de rechtermuisknop, ga naar het menu-item **Subversion** en selecteer **Bijwerken**. Hierdoor worden alle bestanden die bij dit project horen up to date met de repository.
- Als u deze actie wilt beperken tot individuele mappen of bestanden, klap dan de boomstructuur van dit project uit tot het gewenste niveau en klik met de rechtermuisknop op een map of bestandsnaam, ga verder zoals hierboven.



- Als u een of meer bestanden heeft bewerkt, klap dan de boomstructuur van dit project uit tot het niveau waar deze bestanden zich bevinden en klik op de map met de rechtermuisknop. U krijgt nu het menu item **Subversion** waarin u verschillende keuzes heeft. Kies **Compare to base** om de verschillen tussen de versie die u heeft bewerkt en de versie in de repository die u eerder heeft uitgecheckt (De versie 'base'). Het resultaat toont de 'diffs' voor alle bestanden in de map.

- Als u maar een enkel bestand heeft bewerkt dan kan u ook het **Subversion** menu voor dit bestand openen door met de rechtermuisknop op de bestandsnaam in het projectvenster te klikken. Nog makkelijker, u kunt deze menuoptie ook openen door te klikken met de rechtermuisknop in het **Editor** venster waarin u dit bestand heeft geopend.

- Als u een of meerdere bestanden die u heeft bewerkt in wilt checken, klik dan met de rechtermuisknop op een individueel bestand, map of heel project en selecteer **Subversion** → **Vastleggen**. Hierdoor schakelt KDevelop om naar de **Commit** modus, de derde modus naast **Code** en **Debuggen** zoals u kunt zien in de rechterbovenhoek van het hoofdvenster van KDevelop. De afbeelding rechts toont hoe dit eruit ziet. In de **Vastleg** modus, in het bovenste gedeelte zijn de diffs voor de gehele map/project en elk individueel gewijzigd bestand te zien met de wijzigingen gemarkeerd (bekijk de verschillende tabs in dit gedeelte van het venster). Standaard zijn alle gewijzigde bestanden opgenomen in de changeset die u gaat committen, maar u kunt bestanden deselecteren omdat hun wijzigingen geen relatie hebben met waarvoor u een commit wilt uitvoeren. in het voorbeeld rechts heb ik `step-32.cc` en `step-32.prm` gedeselecteerd om de wijzigingen in deze bestanden niets van doen hebben met de andere wijzigingen die ik in dit project heb gemaakt en ik ze nog niet wil inchecken (misschien dat ik dat later alsnog wil doen in een aparte commit). Na een inspectie van de wijzigingen kunt u een commit-bericht in het tekstvak invoeren en op **Vastleggen** rechts drukken om het te versturen.

- Op dezelfde manier als bij het verschil zien, als u een enkel bestand wilt inchecken dan kunt u ook hier weer met rechtermuisknop in het **Editor** venster klikken om het menu item **Subversion** → **Vastleggen** te openen.

Hoofdstuk 9

KDevelop aanpassen

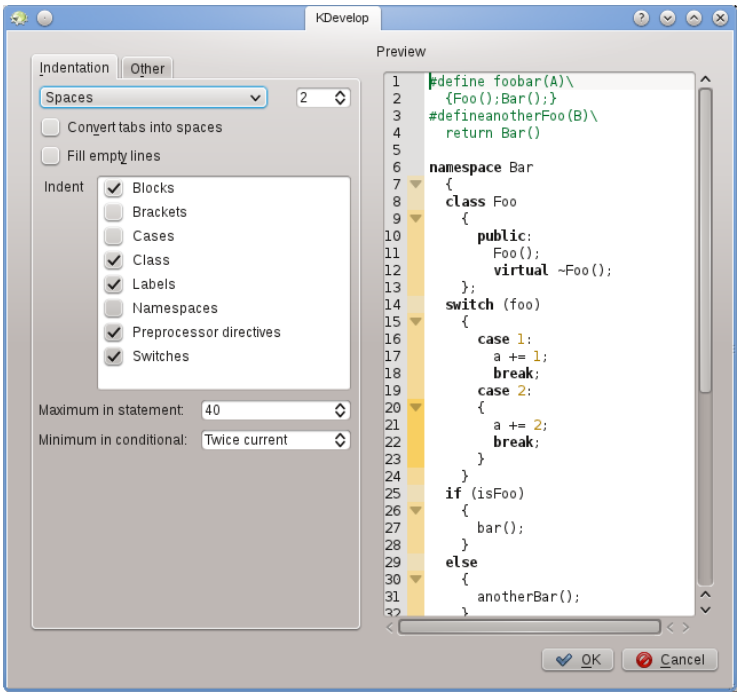
Er zijn momenten dat u het standaard uiterlijk of gedrag van KDevelop wilt aanpassen, bijvoorbeeld omdat u gewend bent aan andere sneltoetsen of omdat uw project voor de broncode een andere inspring-style vereist. In de volgende secties beschrijven we de verschillende manieren hoe u KDevelop aan u behoeften kan aanpassen.

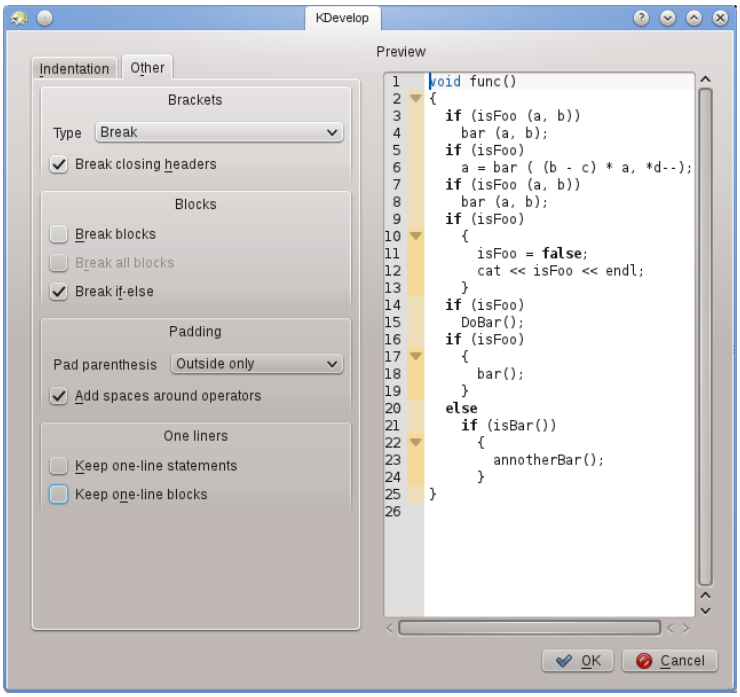
9.1 De editor aanpassen

Er zijn een aantal handige dingen die u kunt instellen rondom de ingebouwde editor van KDevelop. Voor meer algemeen gebruik is het inschakelen van regelnummering via het menu-item **Bewerker** → **Beeld** → **Regelnummering tonen**, zodat het makkelijker is om foutmeldingen van de compiler of debug-berichten in de broncode te lokaliseren. U kunt in hetzelfde menu ook de *Pictogramrand tonen* inschakelen - een kolom links van uw code waar KDevelop door middel van pictogrammen laat zien of er bijvoorbeeld een breakpoint op de geselecteerde regel is.

9.2 Code inspringingen aanpassen

Veel mensen hebben een voorkeur voor een bepaalde stijl waarop de code is opgemaakt. Ook veel projecten hebben een verplichtte stijl voor het inspringen. Die hoeven niet overeen te komen met manier waarop KDevelop inspringt. Maar dit kunt u aanpassen: ga naar het menu-item **Instellingen** → **KDevelop instellen**, klik op **Broncode formatteerprogramma** aan de linkerkant. U kunt een van de veelgebruikte voorgedefinieerde stijlen kiezen, of uw eigen stijl definiëren door een nieuwe stijl aan te maken en deze verder te bewerken. Hiermee creëert u misschien niet exact de stijl voor het inspringen waarin in het verleden de broncode uit uw project is aangemaakt maar u komt er wel dichtbij; u ziet een voorbeeld in de twee afbeeldingen hieronder.


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
#define foobar(A)\
{Foo();Bar();}
#defineanotherFoo(B)\
return Bar()
namespace Bar
{
class Foo
{
public:
Foo();
virtual ~Foo();
};
switch (foo)
{
case 1:
a += 1;
break;
case 2:
{
a += 2;
break;
}
}
if (isFoo)
{
bar();
}
else
{
anotherBar();
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
void func()
{
if (isFoo (a, b))
bar (a, b);
if (isFoo)
a = bar ((b - c) * a, *d--);
if (isFoo (a, b))
bar (a, b);
if (isFoo)
{
isFoo = false;
cat << isFoo << endl;
}
if (isFoo)
DoBar();
if (isFoo)
{
bar();
}
else
if (isBar())
{
annotherBar();
}
}
OK
Cancel

OPMERKING
Met **KDevelop 4.2.2**, kunt u een nieuwe stijl voor een bepaalde mimetype (bijv. voor C++ headerbestanden) aanmaken maar deze stijl komt niet zichtbaar in de lijst met beschikbare stijlen voor andere mimetypes (bijv. voor C++ bronbestanden) ofschoon het natuurlijk handig zou zijn om dezelfde stijl voor beide soorten bestanden te gebruiken. U moet daarom de stijl twee keer aanmaken, Een voor header en een voor bronbestanden. Dit is gemeld als KDevelop bug 272335.

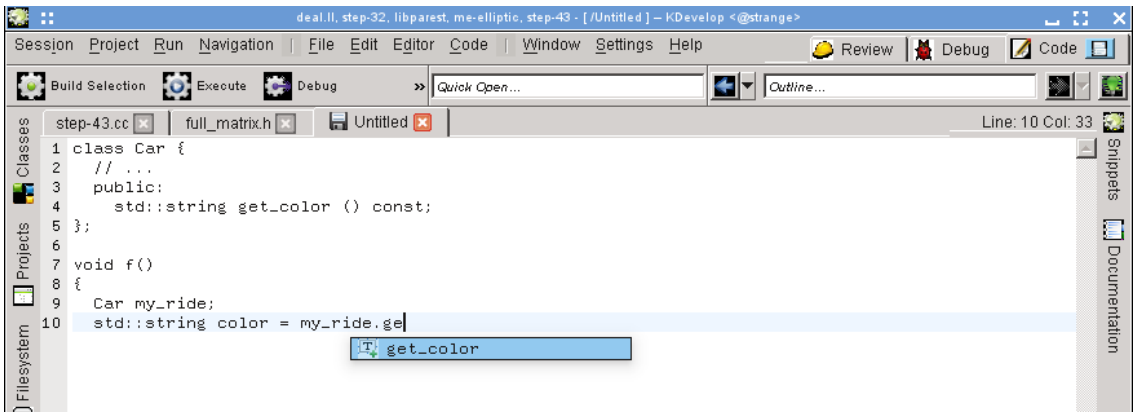
9.3 Sneltoetsen aanpassen

KDevelop heeft een bijna eindeloze lijst van sneltoetsen (enkele zijn genoemd in de 'Secties met handige sneltoetsen' in verschillende hoofdstukken van deze handleiding) die u naar uw smaak kunt aanpassen in het menu-item **Instellingen** → **Sneltoetsen instellen**. Boven in het dialoogvenster kunt u zoeken naar een woord waarna alleen de overeenkomsten zichtbaar zijn; u kunt vervolgens de sneltoetscombinatie aanpassen die met dit commando is verbonden.

Het is gebleken dat het verstandig is om deze twee aan te passen: **Uitlijnen** met de **Tab**-toets (veel mensen voeren meestal niet met de hand tabs in maar geven de voorkeur aan dat de editor de indeling van de broncode verzorgt door de aangepaste sneltoets, hitting **Tab** makes KDevelop inspringt/terugspringt/ de code naar links uitlijnt). De tweede is het in en uitschakelen van **Breekpunt aan/uit** via **Ctrl-B** omdat dit een veel voorkomende handeling is.

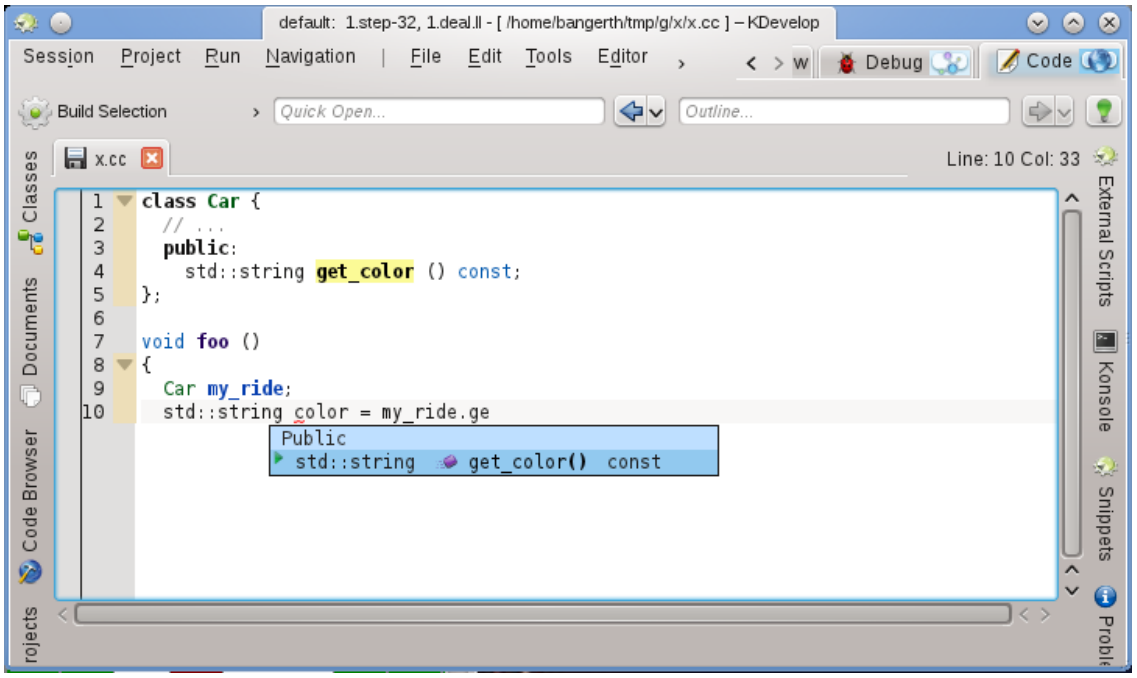
9.4 Automatische aanvulling aanpassen

De beschrijving van automatisch code aanvullen vindt u in deze handleiding voor het schrijven van broncode. In KDevelop, is het afkomstig van twee bronnen: de editor, en het verwerking- en ontleed-onderdeel. De editor (Kate) maakt deel uit van het grote KDE omgeving en biedt automatisch aanvullen aan gebaseerd op in het document voorkomende woorden. Zulke automatische aanvullingen kunt u herkennen aan het voorafgaande icoontje in de tooltip:



Het automatisch aanvullen in de editor kunt u aanpassen via **Instellingen** → **Editor instellen** → **Bewerking** → **Automatische Aanvulling**. U kunt met name instellen hoeveel karakters u moet invoeren voordat automatisch aanvullen een voorstel geeft.

Maar het automatische aanvulling van KDevelop zelf is veel krachtiger omdat het de context in overweging neemt. Het weet bijvoorbeeld welke member functions het moet aanbieden als u `object.` invoert, etc., zoals u hier kunt zien:



De informatie over de context komt van de verschillende plugins die de verschillende talen ondersteunen, deze kunt u pas gebruiken nadat u het bestand voor de eerste keer heeft opgeslagen (omdat dan pas het bestandsselectie gecontroleerd kan worden en de juiste taal ondersteunen).

De automatische aanvulling van KDevelop is zodanig ingesteld dat de aanvullingen zichtbaar worden vanaf het moment dat u begint met typen op vrijwel alle plekken dat een aanvulling toepasselijk zou zijn, Dit kunt u aanpassen in **Instellingen** → **KDevelop instellen** → **Taalondersteuning**. als dit nog niet is ingeschakeld (dit zou standaard wel moeten zijn), zorg er dan voor dat **Automatisch opstarten inschakelen** is ingeschakeld.

KDevelop heeft twee manieren voor het tonen van een aanvulling: **Eenvoudig automatische aanvulling** toont alleen de standaard informatie in de aanvulling-tooltip (bijv. de namespace, klasse, functie, of variabele-naam). Dit zal op de aanvulling van Kate lijken (behalve de pictogrammen).

Maar **Extra informatie** zal ook het type voor elke entry, en in het geval van functies, ook de bijbehorende parameters tonen. Bovendien, als u een parameter aan een functie toevoegt dan zal Automatisch aanvullen een extra info-box boven de cursor tonen over de geselecteerde parameter waaraan u op dat moment werkt.

Automatisch aanvullen van KDevelop zal bovendien zowel in eenvoudige als complete aanvulling bovenaan en groen gemarkeerd de items tonen die overeenkomen met het verwachte type, beter bekend als 'best overeenkomend'.

De drie keuzemogelijkheden in het instellingendialoog voor het niveau van automatisch aanvullen zijn:

- **Nooit**: geeft nooit 'Volledige aanvulling'
- **Indien handmatig opgestart**: Toont alleen 'Extra informatie' als automatisch aanvullen handmatig is opgestart (bijv., wanneer u **Ctrl-Spatie** heeft ingetoetst)
- **Altijd**: geeft altijd 'Volledige aanvulling'

Hoofdstuk 10

Dankbetuigingen en licentie

Documentatie Copyright zie de UserBase KDevelop4/Manual page history

Op- of aanmerkingen over de vertalingen van de toepassing en haar documentatie kunt u melden op http://www.kde.nl/bugs.

Dit document is vertaald in het Nederlands door Freek de Kruijf freekdekruijf@kde.nl.

Dit document is vertaald in het Nederlands door Ronald Stroethoff stroet43@zonnet.nl.

Deze documentatie valt onder de bepalingen van de GNU vrije-documentatie-licentie.