Manuale di KDevelop

Questa documentazione è stata convertita dalla pagina KDevelop4/Manual di KDE UserBase .
Traduzione della documentazione.: Simone Solinas

Indice

Sommario

KDevelop è un ambiente di sviluppo integrato da utilizzare per una grande varietà di attività di programmazione.

Capitolo 1

Cosa è KDevelop?

KDevelop è un ambiente di sviluppo integrato (IDE) moderno per il C++ (e altri linguaggi), questo è una delle tante applicazioni di KDE. Come tale esso viene eseguito su Linux® (anche se si esegue uno degli altri desktop, come GNOME), ma è disponibile per la maggior parte delle altre varianti di UNIX® e anche per Windows.

KDevelop offre tutte le comodità dei moderni IDE. Per progetti e applicazioni di grandi dimensioni, la caratteristica più importante è che KDevelop *interpreta il C++*: analizza l'intero codice sorgente e ricorda le funzioni membro delle classi, dove sono definite le variabili, quali sono i loro tipi, e molte altre cose sul codice. Per esempio, mettiamo il caso che in uno dei file di intestazione del tuo progetto sia dichiarata la seguente classe

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

poi nel tuo programma hai

```
Car my_ride;
// ...do something with this variable...
std::string color = my_ride.ge
```

questo ti avrà ricordato che `my_ride` nell'ultima riga è una variabile di tipo `Car` e ti offre di completare `ge` come `get_color()` poiché questa è l'unica funzione membro della classe `Car` che inizia in questo modo. Invece di continuare a digitare basta premere **Invio** per ottenere la parola completa, questo permette di scrivere di meno, evitare errori di battitura, e non richiede di ricordare i nomi esatti delle centinaia o migliaia di funzioni e classi che compongono i progetti di grandi dimensioni.

Come secondo esempio, supponi di avere un codice come questo:

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

Se passi il mouse sul simbolo `var` nella funzione `bar` hai la possibilità di vedere tutti i modi di utilizzare questo simbolo. Facendo clic su di esso ti verranno mostrati solo i modi di utilizzare questa variabile nella funzione `bar` perché KDevelop sa che la variabile `var` nella funzione `foo` è una variabile diversa. Allo stesso modo, fare clic con il tasto destro sul nome della varibile consente di rinominare la variabile; in questo modo verrà toccata solo la variabile `bar` ma non quella con lo stesso nome presente in `foo`.

Ma KDevelop non è solo un editor intelligente di codice; ci sono altre cose che KDevelop fa' bene. Ovviamente sottolinea il codice sorgente con colori differenti; ha un identatore personalizzabile; ha un interfaccia integrata per il debugger GNU gdb; può mostrare la documentazione per una funzione, se si passa il mouse sopra; può avere a che fare con tipi differenti di ambienti di compilazione e compilatori (ad esempio con **make** e progetti basati su **cmake**), e molte altre cose che saranno discusse in questo manuale.

Capitolo 2

Sessioni e progetti: le basi di KDevelop

In questa sezione, andremo oltre la terminologia di come KDevelop vede il mondo e di come struttura il lavoro. In particolare, introdurremo il concetto di *sessioni* e *progetti* e spiegheremo come è possibile impostare i progetti su cui si desidera lavorare in KDevelop.

2.1 Terminologia

KDevelop ha l'approccio di *sessioni* e *progetti*. Una sessione contiene tutti i progetti che hanno qualcosa a che fare l'uno con l'altro. Per gli esempi che seguono, presupponiamo che tu sia lo sviluppatore sia di una libreria che di un'applicazione che la utilizza. Puoi pensare alle librerie base di KDE come se fossero la prima e a KDevelop come se fosse l'ultima. Un altro esempio: diciamo che sei un hacker del kernel Linux® ma che stai anche lavorando su un driver per un dispositivo per Linux® che non è stato ancora aggiunto all'albero dei sorgenti del kernel.

Quindi, prendendo quest'ultimo come esempio, avresti una sessione in KDevelop che ha due progetti: il kernel Linux® e il driver del dispositivo. Vorrai raggrupparli in una singola sessione (piuttosto che avere due sessioni con un singolo progetto ciascuno) perché sarà utile vedere le funzioni del kernel e le strutture dati in KDevelop ogni volta che scrivi il codice sorgente per il driver — per esempio in modo da ottenere le funzioni del kernel e i nomi delle variabili auto-espanse, o in modo da vedere la documentazione delle funzioni del kernel mentre si agisce sul driver del dispositivo.

Ora immagina anche di essere un sviluppatore di KDE. Quindi avresti una seconda sessione che contiene KDE come progetto. Potresti, in linea di principio avere una sola sessione per tutto questo, ma non c'è una vera ragione per questo: nel tuo lavoro per KDE, non hai bisogno di accedere alle funzioni del kernel o ai driver del dispositivo; e non vuoi che i nomi delle classi di KDE siano espansi automaticamente mentre lavori sul kernel Linux®. Infine, la compilazione di alcune librerie di KDE è indipendente dal ricompilare il kernel Linux® (invece ogni volta che compili il driver del dispositivo sarebbe anche bene ricompilare il kernel Linux® se alcuni dei file di intestazione del kernel sono cambiati).

Infine, un altro uso per le sessioni è se lavori sia sulla versione di sviluppo di un progetto, che su un altro ramo: in questo caso,non vuoi che KDevelop confonda le classi che appartengono alla linea principale e al ramo, quindi avresti due sessioni, con lo stesso insieme di progetti, ma da diverse cartelle (che corrispondono a diversi rami di sviluppo).

2.2 Impostare una sessione e importare un progetto esistente

Rimanendo all'esempio del kernel Linux® e dei driver — potresti voler sostituire i tuoi progetti o librerie di questi due esempi. Per creare una nuova sessione che contenga questi due progetti vai nel menu in alto a sinistra **Sessione** → **Avvia una nuova sessione** (o se è la prima volta che usi KDevelop: è sufficiente usare la sessione predefinita, quella del primo utilizzo, che è vuota).

Poi vogliamo popolare questa sessione con dei progetti che per il momento assumiamo già esistenti da qualche parte (il caso della creazione da zero di progetti è discusso altrove in questo manuale). Per fare questo, esistono essenzialmente due metodi, a seconda che il progetto sia già da qualche parte sul tuo disco fisso o se deve essere scaricato da un server.

2.2.1 Opzione 1: importare un progetto da un server vcs

Assumendo che il progetto che vogliamo creare — il kernel Linux® — si trovi in qualche sistema di versione controllo su un server, che non hai ancora scaricato sul tuo disco fisso. In questo caso, vai nel menu **Progetto** per creare il kernel Linux® come progetto nella sessione corrente e poi segui questi passi:

- Vai in **Progetto** → **Preleva progetto** per importare un progetto
- Quindi hai diverse opzioni per avviare un nuovo progetto nella sessione corrente, a seconda della provenienza dei file sorgente: puoi far puntare KDevelop solo ad una cartella che esiste (vedi l'opzione 2 sotto), oppure puoi chiedere a KDevelop di ottenere i sorgenti da un deposito.
- Assumendo che tu non ne abbia già prelevata una versione:
  - Nella finestra di dialogo, sotto **Seleziona il sorgente**, scegliere di usare **Dal file system**, **Subversion**, **Git**, **GitHub** o **KDE**
  - Scegli una cartella di lavoro come destinazione nella quale dovrebbero essere posti i sorgenti prelevati
  - Scegli un URL per la posizione del deposito da cui i file sorgente possono essere ottenuti
  - Premi **Ottieni**. Questa operazione può richiedere parecchio tempo; a seconda della velocità della tua connessione e dalla dimensione del progetto. Purtroppo, in KDevelop 4.2.x la barra di avanzamento in realtà non mostra nulla, ma si può tenere traccia dei progressi guardando periodicamente l'output della riga di comando

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

    per vedere quanti dati sono stati già scaricati.

> NOTA
> Il problema con la barra di avanzamento è stato segnalato nel bug di KDevelop 256832

> NOTA
> In questo processo, ho anche ottenuto il messaggio d'errore *È necessario specificare un percorso valido per il progetto* che può essere tranquillamente ignorato.

- Ti chiede di selezionare in questa cartella un file di progetto di KDevelop. Dal momento che probabilmente non ne hai ancora uno, basta premere **Successivo**

- Premi ancora **Successivo**
- poi KDevelop ti chiederà di scegliere un gestore per il progetto. Se questo progetto usa dei make file standard di UNIX® scegli gestore progetto Makefile personalizzato
- KDevelop quindi inizierà ad analizzare l'intero progetto. Anche in questo caso, ci vorrà un bel po' per completare tutti i file e le classi ecc.. In basso a destra della finestra principale, c'è una barra di avanzamento che mostra quanto tempo dura questo processo. (Se disponi di un processore multicore, questo processo può essere accelerato andando alla voce del menu **Impostazioni** → **Configura KDevelop** , poi selezionando **Analizzatore in background** sulla sinistra, e aumentando il numero di thread per l'analisi in background sulla destra).

2.2.2 Opzione 2: importare un progetto che è già sul tuo disco fisso

In alternativa, se il progetto sul quale vuoi lavorare esiste già sul tuo disco fisso (per esempio, perché lo hai scaricato come un file tar da un server FTP, perché hai già prelevato una versione del progetto da un sistema di controllo versione, o perché è il tuo progetto ed esiste *solo* sul tuo disco fisso), allora usa **Progetto** → **Apri/Importa progetto** e nella finestra di dialogo scegli la cartella in cui risiede il progetto.

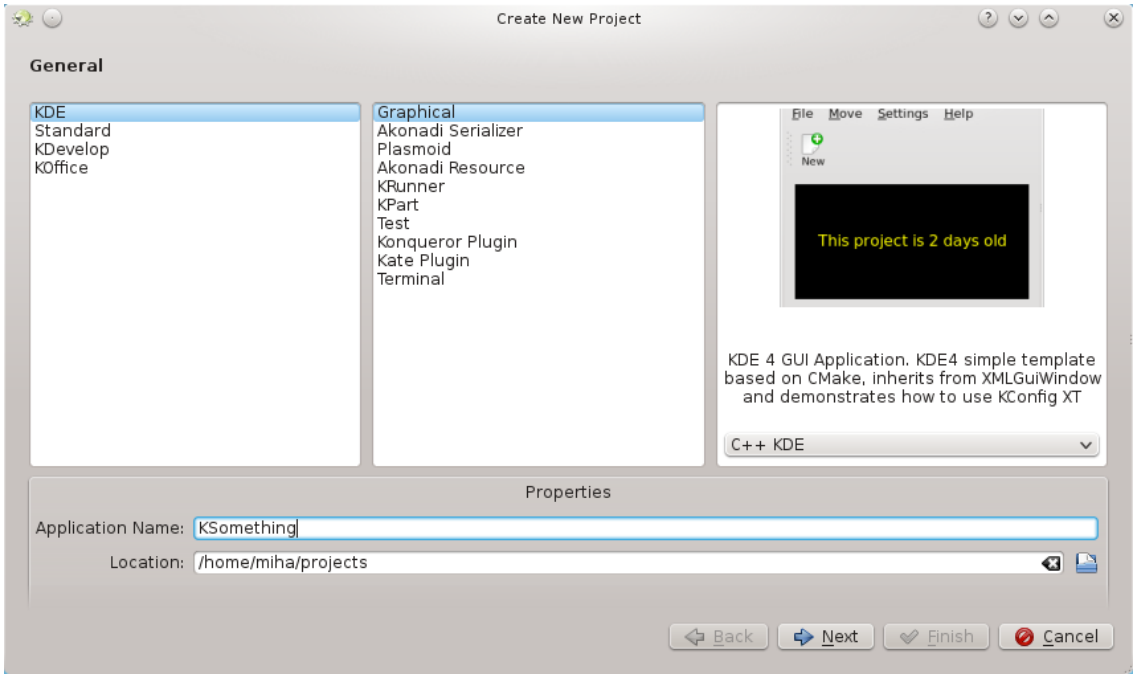
2.3 Impostare un'applicazione come secondo progetto

La prossima cosa che vuoi fare è impostare altri progetti nella stessa sessione. Nell'esempio precedente, volevi aggiungere come secondo progetto il driver del dispositivo, che puoi fare eseguendo gli stessi passi.

Se hai applicazioni o librerie multiple, ripeti semplicemente i passi per aggiungere sempre più progetti alla tua sessione.

2.4 Creare progetti da zero

C'è, naturalmente, anche la possibilità che tu voglia iniziare un nuovo progetto da zero. Questo può essere fatto usando la voce del menu **Progetto** → **Nuovo da modello...** che ti si presenta con una finestra per la scelta del modello. Alcuni modelli di progetto sono forniti da KDevelop, ma ce ne sono molti altri se si installa l'applicazione KAppTemplate. Scegli il tipo di progetto e il linguaggio di programmazione dalla finestra di dialogo, immetti un nome e una posizione per il tuo progetto e fai clic su **Successivo**.



La seconda pagina della finestra di dialogo ti permette di impostare un sistema di controllo versione. Scegli il sistema che vuoi usare, e completa se necessario la configurazione specifica del sistema. Se non vuoi usare un sistema di controllo versione, o vuoi impostarlo manualmente in seguito, scegli **Nessuno**. Quando sei contento delle tue scelte, premi **Completa**.

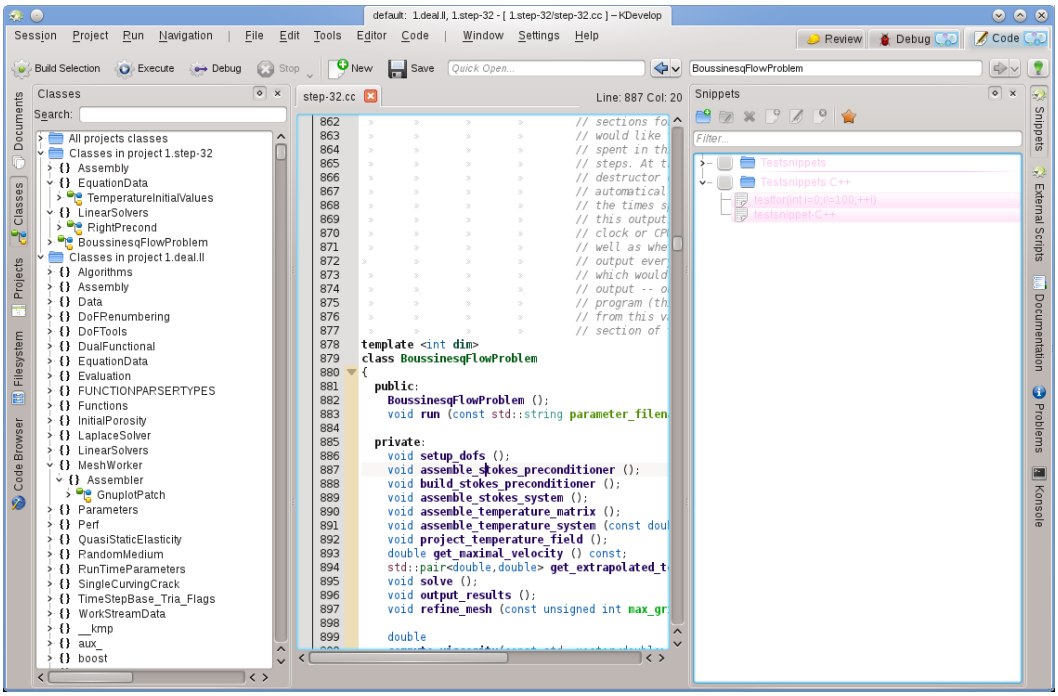
Ora il tuo progetto è stato creato, per cui puoi provare a compilarlo o a installarlo. Alcuni modelli conterranno dei commenti nel codice, o pure un file README separato, e si raccomanda di leggerlo prima. Poi puoi iniziare a lavorare sul tuo progetto, aggiungendo qualsiasi funzionalità desideri.

Capitolo 3

Lavorare con il codice sorgente

Oltre a fare il debug, quello su cui passerai la maggior parte del tempo quando svilupperai software sarà leggere e scrivere codice sorgente. A tal fine, KDevelop ti offre molti modi diversi per analizzare il codice sorgente e per rendere più produttiva la scrittura. Come discusso più in dettaglio nelle sezioni seguenti, KDevelop non è solo un editor di sorgenti — piuttosto, si tratta di un sistema di gestione dei sorgenti che ti da diversi modi di vedere le informazioni estratte dai file che insieme compongono il codice sorgente della sessione.

3.1 Strumenti e viste



Al fine di lavorare con i progetti, KDevelop possiede il concetto degli *strumenti*. Uno strumento fornisce una visione particolare del sorgente, o di un'azione che può essere fatta con esso. Gli strumenti sono rappresentati da pulsanti nella finestra (nel testo verticale lungo i margini destro e sinistro, o in orizzontale lungo il margine inferiore). Se fai clic su di essi, si espandono in una

sotto finestra — una *vista* — all'interno della finestra principale; se fai ancora clic sul pulsante dello strumento, la sottofinestra scompare di nuovo.

Per far chiudere una sottofinestra, puoi fare clic sulla **x** in alto a destra della sottofinestra.

L'immagine sopra mostra una particolare selezione di strumenti, allineata a sinistra e al margine destro, nella foto, lo strumento **Classi** è aperto sulla sinistra e lo strumento **Frammenti** lo è sulla destra, insieme all'editor del file sorgente nel mezzo. In pratica, per la maggior parte del tempo avrai solo l'editor e forse lo strumento **Classi** o quello **Browser del codice** aperti sulla sinistra. Gli altri strumenti si apriranno solo temporaneamente durante l'uso, lasciando per la maggior parte del tempo più spazio all'editor.

Quando esegui KDevelop la prima volta, dovresti avere già il pulsante dello strumento **Progetti**. Fai clic su di esso: si aprirà una sottofinestra che mostra, in basso, i progetti che hai aggiunto alla sessione, e una vista, in alto, del file system delle cartelle dei tuoi progetti.

Ci sono molti altri strumenti che puoi usare con KDevelop non tutti sono presenti all'inizio come pulsanti sulla finistra. Per aggiungerne altri, vai alla voce del menu **Finestra** → **Aggiungi vista strumento**. Qui ci sono alcune viste che probabilmente troverai utili:

- **Classi**: un elenco completo di tutte le classi che sono definite in uno dei progetti o la tua sessione con tutte le sue funzioni membro e variabili. Facendo clic su uno qualsiasi dei membri si apre una finestra dell'editor del sorgente relativa all'elemento sul quale hai fatto clic.
- **Documenti**: elenca alcuni dei file visitati più di recente, per tipo (ad esempio sorgenti, patch, documenti di testo).
- **Browser del codice**: a seconda della posizione del cursore in un file, questo strumento mostra le cose che sono in relazione. Per esempio, se sei su una riga `#include`, vengono mostrate informazioni sul file che stai includendo ad esempio quali classi sono dichiarate in questo file, se sei su una riga vuota nell'ambito del file, vengono mostrate le classi e le funzioni dichiarate e definite nel file (tutte come collegamenti: se fai clic su una di esse questo ti porterà al punto del file in cui è in realtà la dichiarazione o la definizione); se sei nella definizione di una funzione, questo ti mostrerà dov'è la dichiarazione e ti offrirà un elenco di posti in cui viene usata la funzione.
- **Filesystem**: ti mostra una vista ad albero del file system.
- **Documentazione**: ti permette di cercare pagine man e altra documentazione d'aiuto.
- **Frammenti**: fornisce le sequenze di testo che si usano di più e che non si vuole scrivere ogni volta. Ad esempio, nel progetto dalla quale l'immagine qui sopra è stata creata, vi è una frequente necessità di scrivere del codice del tipo

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

Questa è un'espressione complicata ma sembrerà sempre la stessa ogni volta che avrai bisogno di un ciclo simile — cosa che la rende una buona candidata per un frammento.

- **Konsole**: apre una finestra della riga di comando nella finestra principale di KDevelop, per il comando occasionale che potresti inserire (ad es. per eseguire `./configure`).

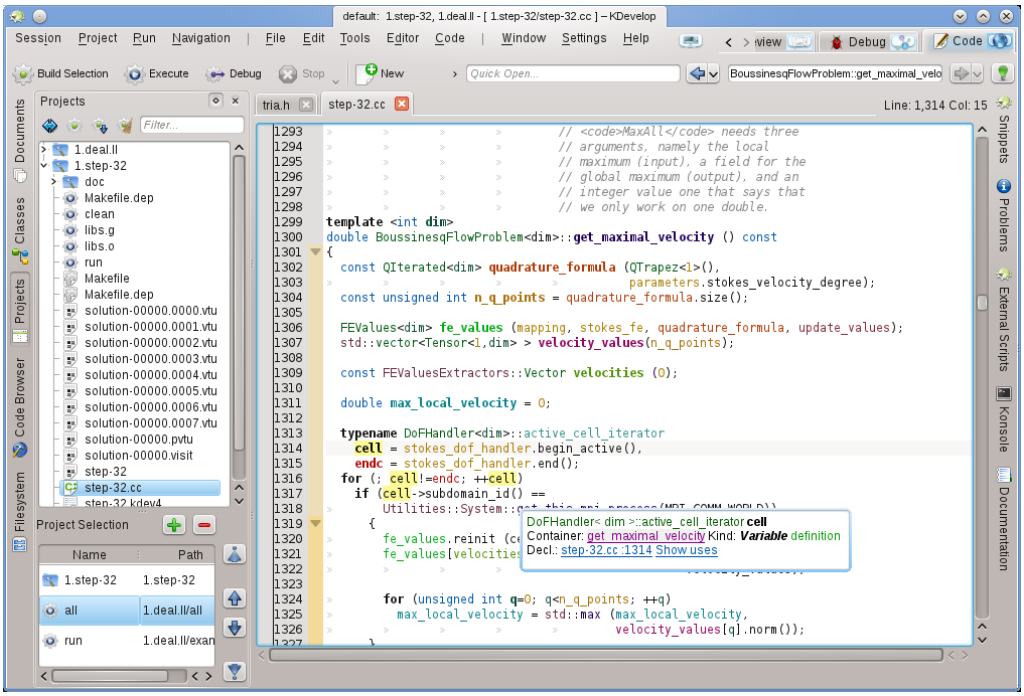
Qui puoi trovare un elenco completo di strumenti e viste.

Per molti programmatori, lo spazio verticale dello schermo è il più importante. A questo proposito, è possibile organizzare le tue viste strumento sul margine sinistro e destro della finestra: per spostare uno strumento, fare clic sul simbolo con il tasto destro del mouse e selezionare una nuova posizione.

3.2 Esplorare il codice sorgente

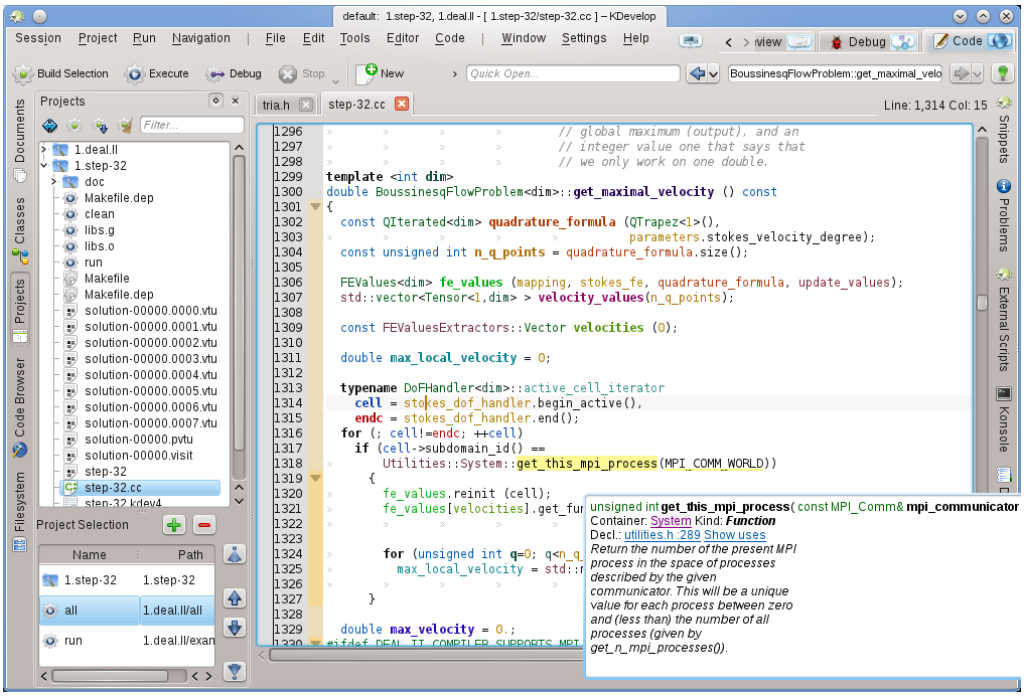
3.2.1 Informazioni locale

KDevelop *interpreta* il codice sorgente, e di conseguenza è veramente valido nel fornire informazioni sulle variabili o funzioni che possono apparire nel tuo programma. Per esempio, ecco un'istantanea del lavoro su un pezzo di codice e sul passaggio del mouse sopra il simbolo `cell` nella riga 1316 (se stai lavorando usando solo la tastiera, è possibile ottenere lo stesso effetto tenendo premuto per un po' il tasto **Alt**):



KDevelop mi mostra un suggerimento che include il tipo della variabile (qui: `DoFHandler<dim> active_cell_iterator`), dove è dichiarata questa variabile (il *contenitore*, che qui è la funzione circondata `get_maximal_velocity` dal momento che è una variabile locale), cosa è (una variabile, non una funzione, classe o namespace) e dove è dichiarata (nella riga 1314, solo poche righe sopra).

Nel contesto attuale, il simbolo sul quale viene passato il mouse non ha alcuna documentazione associata. In questo esempio, il mouse viene fatto passare sopra il simbolo `get_this_mpi_process` nella riga 1318, il risultato è questo:

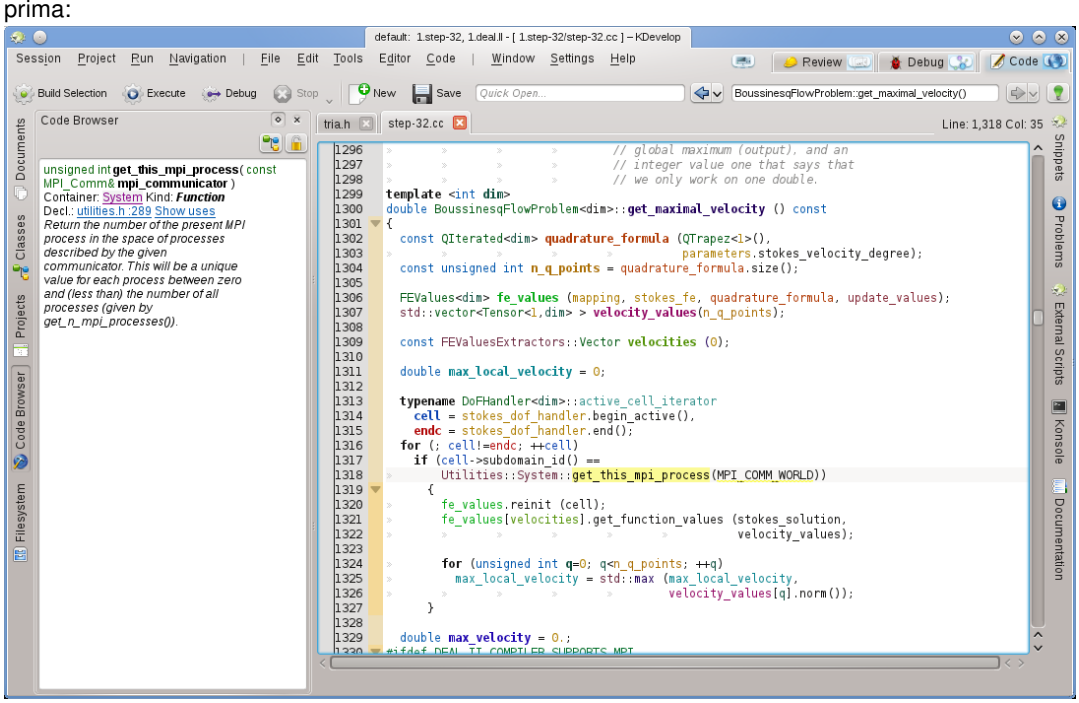


Qui, KDevelop ha fatto un riferimento incrociato della dichiarazione da un file completamente diverso (`utilities.h`, che in realtà sta anche in un diverso progetto della stessa sessione) con il commento in stile doxygen che accompagna la dichiarazione.

Ciò che rende questi suggerimenti ancora più utili è il fatto che sono dinamici: posso fare clic sul contenitore per ottenere informazioni sul contesto nel quale la variabile è dichiarata (cioè sul namespace di `System`, come per esempio dove è dichiarato, definito, usato, o com'è la sua documentazione) e posso fare clic sui collegamenti di color blu che consentono di resettare la posizione del cursore alla posizione della dichiarazione del simbolo (ad es. in `utilities.h`, riga 289) o darmi un elenco di posti in cui questo simbolo viene utilizzato nel file corrente o in tutti i progetti della sessione corrente. Quest'ultimo punto è spesso utile se vuoi analizzare come, per esempio, è utilizzata una funzione particolare in codici di grandi dimensioni.

NOTA

Le informazioni contenute in un suggerimento sono rapide - dipende da te tenendo premuto il tasto **Alt** o passando il mouse. Se vuoi un posto permanente per queste, apri lo strumento **Browser del codice** in una delle sotto finestre. Per esempio, qui il cursore si trova sulla stessa funzione dell'esempio precedente, e la vista strumento a sinistra presenta lo stesso tipo di informazione del suggerimento di prima:



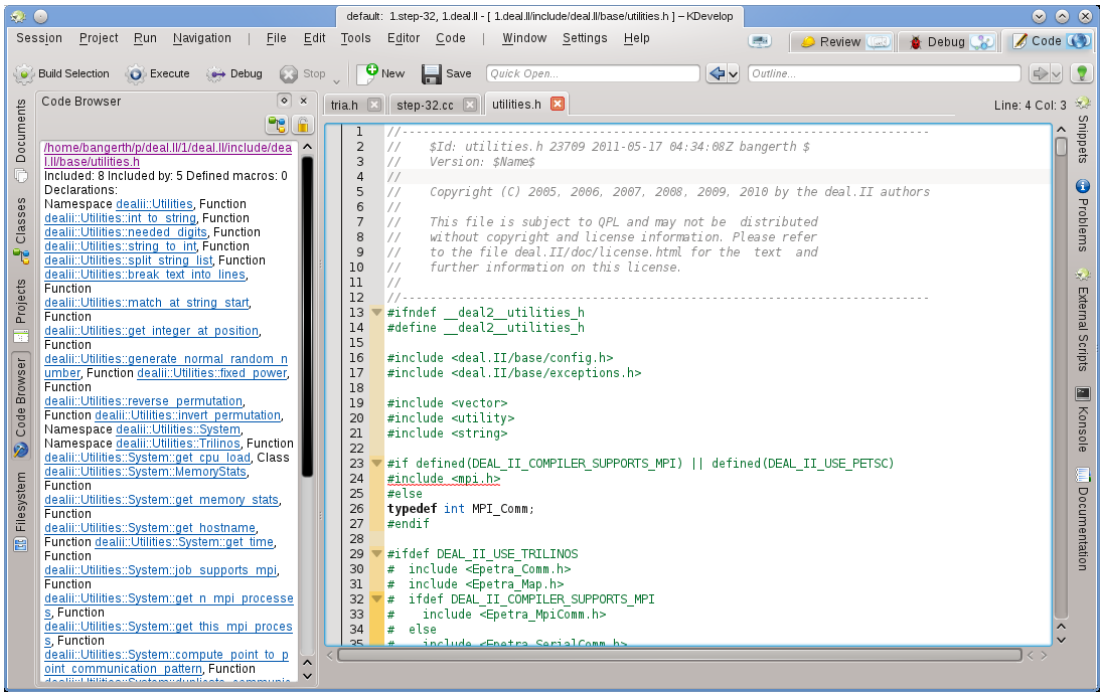
Spostando il cursore a destra cambiano le informazioni presentate sulla sinistra. Inoltre, facendo clic sul pulsante **Blocca la vista corrente** in alto a destra questo permette di bloccare queste informazioni, rendendole indipendenti dal movimento del cursore mentre analizzi l'informazione fornita.

NOTA

Questo tipo di informazioni di contesto è disponibile in molti altri posti inKDevelop, non solo nell'editor di sorgente. Ad esempio, tenendo premuto **Alt** in un elenco di completamento (ad es. quando si esegue un'apertura rapida) fornisce anche informazioni di contesto relative al simbolo corrente.

3.2.2 Informazioni sul ambito del file

Il passo successivo è quello di ottenere informazioni su tutto il file sorgente a cui stai lavorando in questo momento. A tal fine, posiziona il cursore nell'ambito del file nel file corrente e guarda come viene mostrato lo strumento **Browser del codice**:

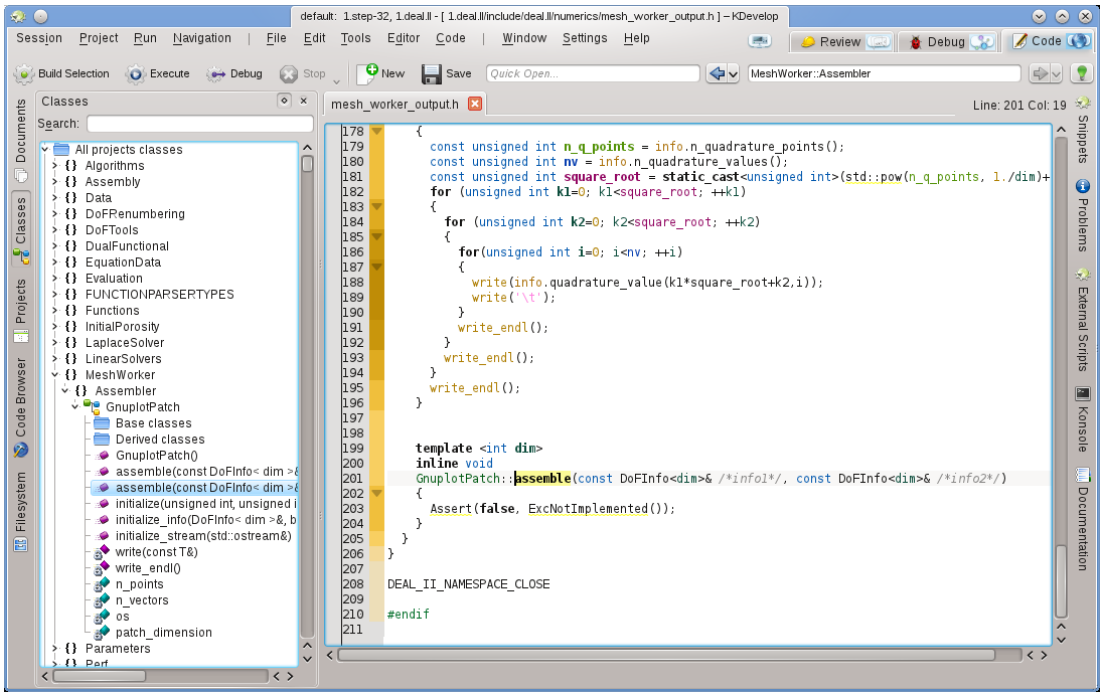


Qui sono mostrati un elenco di namespace, classi e funzioni dichiarate o definite nel file, offrendoti una panoramica di quello che sta succedendo in questo file e un mezzo per passare direttamente ad una qualsiasi di queste dichiarazioni o definizioni senza andare da un punto all'altro del file o alla ricerca di un simbolo particolare.

> NOTA
> Le informazioni visualizzate dell'ambito del file sono le stesse di quelle presentate nella modalità 'Schema riassuntivo' discusse di seguito per la navigazione del codice sorgente, la differenza è che la modalità schema riassuntivo rappresenta solo un suggerimento temporaneo.

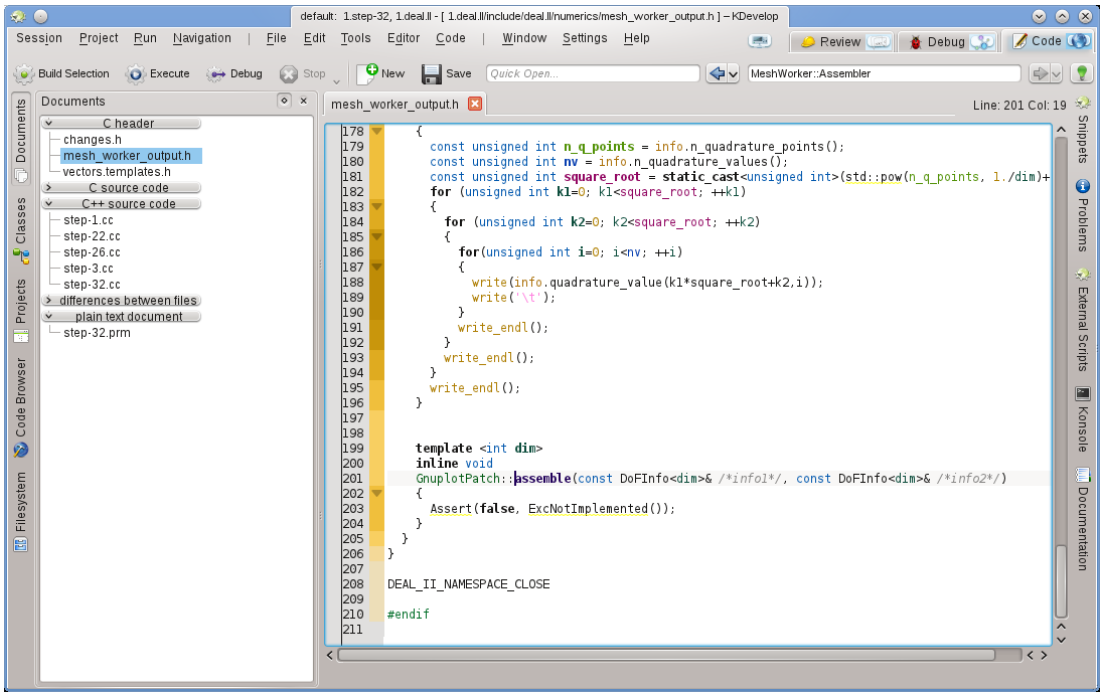
3.2.3 Informazioni sull'ambito del progetto e della sessione

Ci sono molti modi per ottenere informazioni sull'intero progetto (o, di fatto, su tutti i progetti in una sessione). Questo tipo di informazione è tipicamente fornita attraverso vari strumenti vista. Ad esempio, lo strumento **Classi** fornisce una struttura ad albero di tutte le classi e circonda i namespace di tutti i progetti in una sessione, insieme con le funzioni membro e le variabili membro di ciascuna di queste classi:



Passando il mouse sopra una voce vengono fornite ulteriori informazioni sul simbolo, la posizione della sua dichiarazione e definizione, e i suoi usi. Facendo doppio clic su una voce in questa vista ad albero si apre una finestra nella posizione in cui il simbolo è dichiarato o definito.

Ma ci sono altri modi di vedere le informazioni globali. Ad esempio, lo strumento **Documenti** fornisce una vista di un progetto in termini di tipi di file o altri documenti di cui è composto questo progetto:



3.2.4 Spiegazione dei colori di evidenziazione

KDevelop usa una varietà di colori per evidenziare diversi oggetti nel codice sorgente. Se sai cosa significano i diversi colori, è possibile estrarre rapidamente molte informazioni dal codice sorgente solo guardando i colori, senza leggere un singolo carattere. Le regole di evidenziazione sono le seguenti:

- Gli oggetti di tipo Class / Struct, Enum (i valori e il tipo), funzioni (globali), e i membri della classe hanno ciascuno un proprio colore assegnato (le classi sono verdi, le enumerazioni sono di colore rosso scuro, ed i membri sono di colore giallo scuro o viola, le funzioni (globali) sono sempre viola).
- Tutte le variabili globali sono colorate di verde scuro.
- Gli identificatori che sono dei typedef di altri tipi sono colorati di color foglia di tè.
- Tutte le dichiarazioni e definizioni degli oggetti sono in neretto.
- Se si accede ad un membro dall'interno del contesto dove è definito (classe di base o derivata) appare in giallo, altrimenti appare in viola.
- Se un membro è privato o protetto, si colora di un colore leggermente più scuro quando viene usato.
- Per le variabili locali nell'ambito del corpo della funzione, i colori sono scelti in base al hash della identificatore. Questo include i parametri della funzione. Un identificatore avrà sempre lo stesso colore nel suo ambito (ma lo stesso identificatore otterrà un colore diverso se rappresenta un oggetto diverso, cioè se viene ridefinito in un ambito nidificato), e di solito otterrai lo stesso colore per lo stesso nome dell'identificatore in ambiti diversi. Quindi, se disponi di molteplici funzioni che assumono gli stessi nomi per gli identificatori, tutti gli argomenti avranno lo stesso aspetto in termini di colore. Questi colori possono essere disattivati separatamente dalla colorazione globale nella finestra delle impostazioni.
- Gli indentificatori per i quali KDevelop non può determinare la dichiarazione sono colorati in bianco. Questo talvolta può essere causato dalla perdita delle direttive `#include`.
- In aggiunta a questa colorazione, sarà applicata la normale sintassi di evidenziazione dell'editor, come noto da Kate. La semantica di evidenziazione di KDevelop sovrascriverà sempre l'evidenziazione dell'editor in presenza di un conflitto.

3.3 Navigare nel codice sorgente

Nella sezione precedente, abbiamo discusso dell'esplorazione del codice sorgente, cioè ottenere informazioni sui simboli, sui file e progetti. Il passo successivo consisterà nel saltare da un punto all'altro del codice, cioè navigarlo. Ci sono ancora diversi livelli ai quali questo può essere fatto: locale, dentro un file, e all'interno di un progetto.

> NOTA
> Molti dei modi di attraversare il codice sono accessibili dal menu **Navigazione** nella finestra principale di KDevelop.

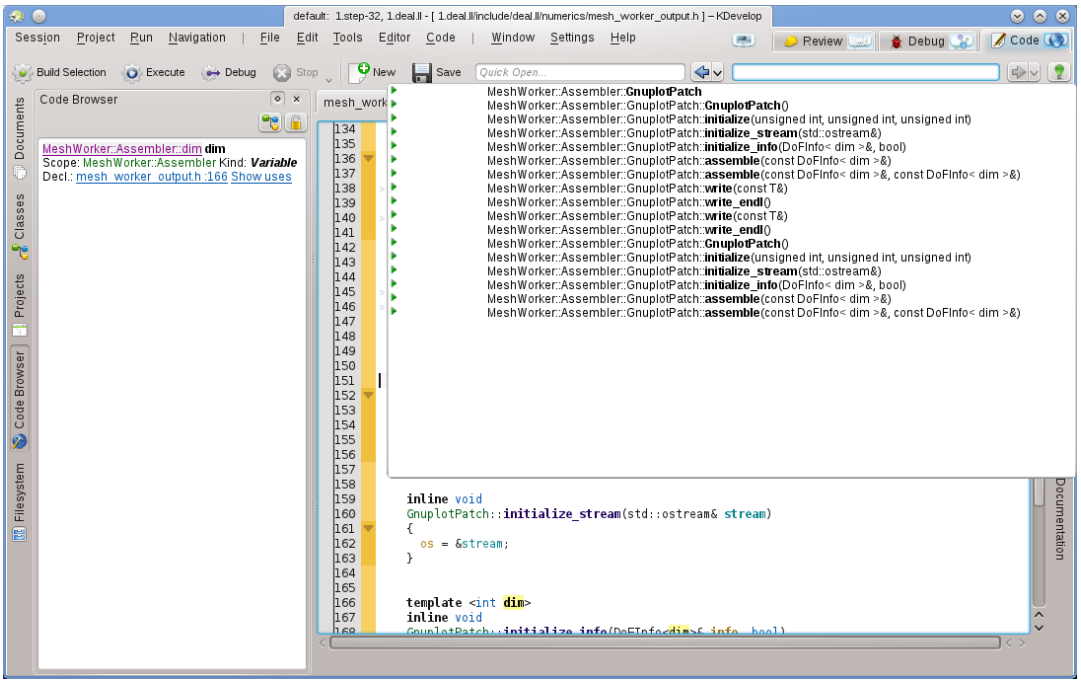
3.3.1 Navigazione locale

KDevelop è molto di più di un editor, ma è *anche* un editor di sorgenti. Come tale, ovviamente puoi spostare il cursore verso l'alto, il basso, a sinistra o a destra in un file sorgente. Puoi anche usare i tasti **Pag↑** e **Pag↓**, e tutti gli altri comandi che usi da qualsiasi editor utile.

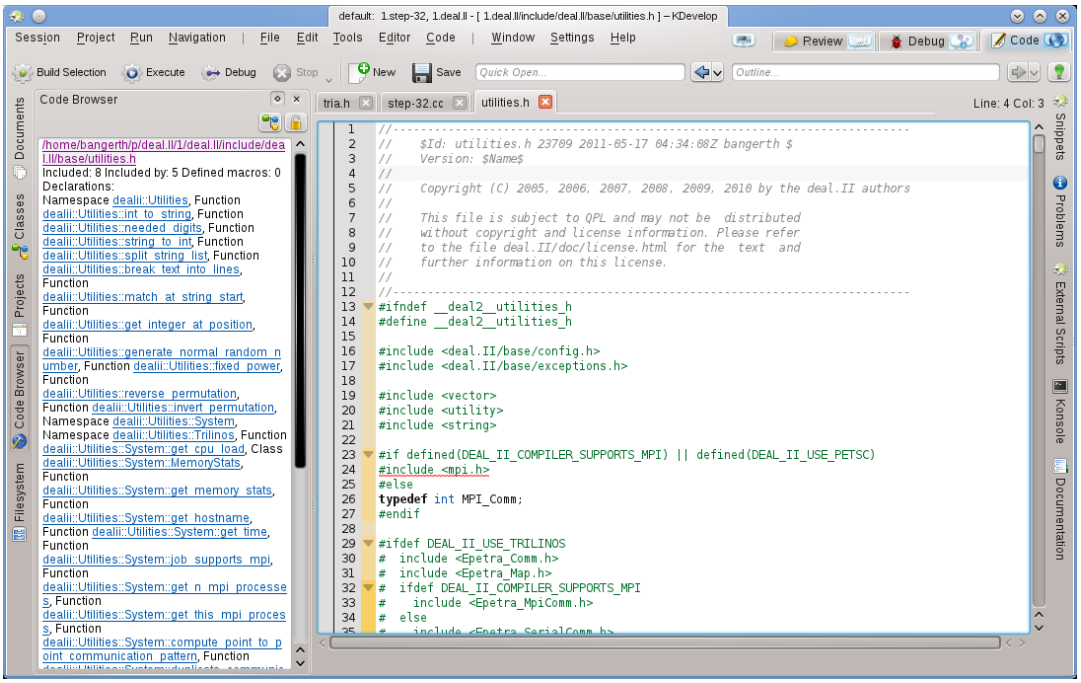
3.3.2 Navigazione dell'ambito del file e modalità schema riassuntivo

All'ambito del file, KDevelop offre molti modi diversi di attraversare il codice sorgente. Per esempio:

- **Schema riassuntivo**: puoi ottenere uno schema riassuntivo di quello che c'è nel file corrente in almeno tre modi diversi:
  - Facendo clic nella casella di testo **Schema riassuntivo** in alto a destra della finestra principale, o premendo **Alt-Ctrl-N** si apre un menu a discesa che elenca le dichiarazioni di tutte le funzioni e classi:



  Poi potrai selezionare quale scegliere, o — se ce ne sono molte — iniziare a digitare del testo che potrebbe apparire nei nomi mostrati; in questo caso, mentre continui a digitare, l'elenco diventa sempre più piccolo mentre i nomi che non corrispondono a quello che hai già digitato vengono rimossi finché non sei pronto a selezionarne uno.
  - Selezionando il cursore nell'ambito del file (cioè fuori da qualsiasi dichiarazione o definizione di funzione o classe) e avendo aperto lo strumento **Browser del codice**:



Questo fornisce anche uno schema riassuntivo di quello che sta accadendo nel file corrente, e ti permette di selezionare il punto in cui vuoi andare.

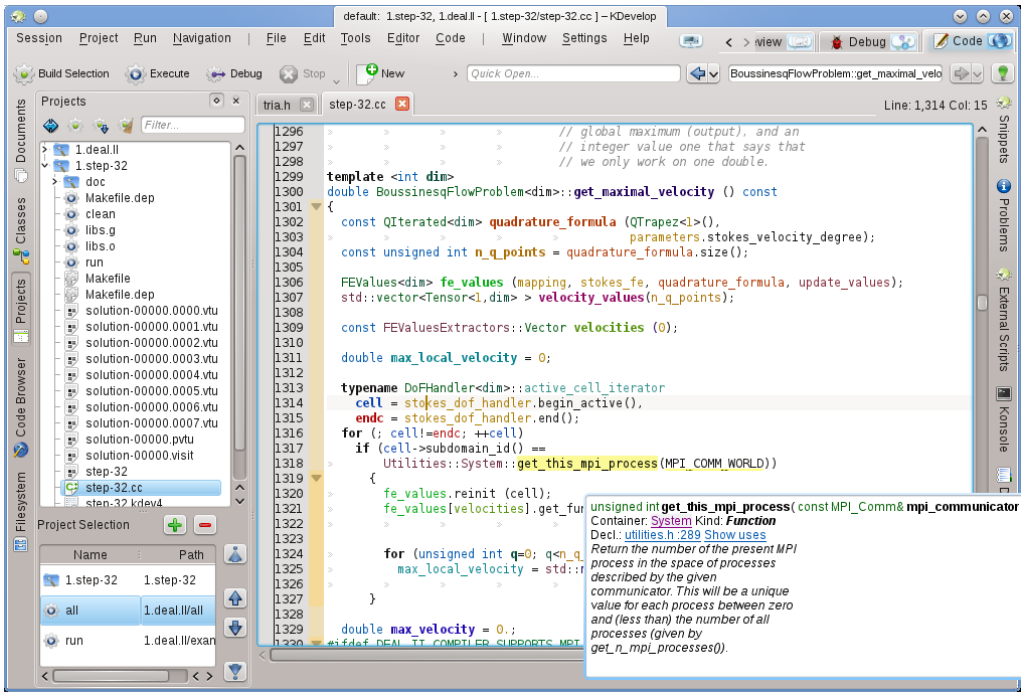
  – Passando il mouse sopra la scheda su uno dei file aperti viene prodotto anche uno schema riassuntivo del file nella scheda.

- I file sorgente sono organizzati come un'elenco di dichiarazioni e definizioni. Premendo **Alt-Ctrl-PgUp** e **Alt-Ctrl-PgDown** si salta alla definizione di funzione precedente o successiva in questo file.

3.3.3 Navigazione dell'ambito della sessione e del progetto: navigazione semantica

Come detto da altre parti, KDevelop di solito non considera file sorgente individuali ma piuttosto guarda nel suo insieme ai progetti (o, piuttosto, a tutti i progetti che sono parte della sessione corrente). Di conseguenza, offre molti modi di attraversare interi progetti. Alcuni di questi sono derivati da quello che abbiamo già discusso nella sezione Esplorare il codice sorgente mentre altri sono molto diversi. Il tema comune è che queste caratteristiche di navigazione sono basate sulla *comprensione semantica* del codice, cioè ti offrono qualcosa che richiede l'analisi di progetti interi e dei dati collegati.

- Come hai visto nella sezione Esplorare il codice sorgente, puoi ottenere un suggerimento che spiega namespace, classi, funzioni o nomi di variabili passando sopra il tuo mouse o tenendo premuto per un po' il tasto **Alt**. Ecco un esempio:



Facendo clic sui collegamenti delle dichiarazioni di un simbolo o espandendo l'elenco degli usi questo consente di saltare in queste posizioni. Un effetto simile può essere ottenuto utilizzando lo strumento **Browser del codice** discusso anche in precedenza.

- Un modo più veloce per passare alla dichiarazione di un simbolo senza dover fare clic sui collegamenti nel suggerimento consiste nell'abilitare temporaneamente la **Modalità navigazione dei sorgenti** tendendo premuti i tasti **Alt** e **Ctrl**. In questo modo sarà possibile fare clic direttamente su qualsiasi simbolo nell'editor per passare alla sua dichiarazione.

- **Apertura veloce**: un modo molto potente di saltare ad altri file o posizioni è usare i vari metodi di *apertura rapida* in KDevelop. Ci sono quattro versioni di questi:

  – **Apertura veloce classe** (**Navigazione** → **Apertura veloce classe** o **Alt-Ctrl-C**): otterrai un elenco di tutte le classi in questa sessione. Inizia a digitare il nome (una parte) di una classe e l'elenco continuerà a ridursi a soltanto quelle classe che corrispondono a quello che hai scritto finora. Se l'elenco è abbastanza corto, seleziona un elemento usando i tasti su e giu e KDevelop ti porterà dov'è dichiarata la classe.

  – **Apertura veloce funzione** (**Navigazione** → **Apertura veloce funzione** o **Alt-Ctrl-M**): otterrai un elenco di tutte le funzioni (membro) che fanno parte del progetto in questa sessione, che puoi selezionare nello stesso modo visto sopra. Nota che questo elenco potrebbe includere sia la definizione della funzione che la dichiarazione della stessa.

  – **Apertura veloce file** (**Navigazione** → **Apertura veloce file** o **Alt-Ctrl-O**): otterrai un elenco di tutti i file che sono parte dei progetti di questa sessione, che puoi selezionare nello stesso modo visto qui sopra.

  – **Apertura veloce universale** (**Navigazione** → **Apertura veloce** o **Alt-Ctrl-Q**): se ti dimentichi quale combinazione di tasti è associata ai comandi visti sopra, questo è il coltellino svizzero — che semplicemente ti presenta un elenco combinato di tutti i file, funzioni, classi e altre cose che hai selezionato.

- **Salta alla dichiarazione/definizione**: quando si implementa una funzione (membro), c'è spesso bisogno di ritornare al punto in cui è dichiarata la funzione, per esempio per tenere sincronizzato l'elenco degli argomenti delle funzioni tra dichiarazioni e definizioni, o per aggiornare la documentazione. Per farlo, metti il cursore sul nome della funzione e seleziona **Navigazione** → **Salta alla dichiarazione** (o premi **Ctrl-.**) per raggiungere il posto in cui è dichiarata la funzione. Ci sono diversi modi per tornare dove si era prima:

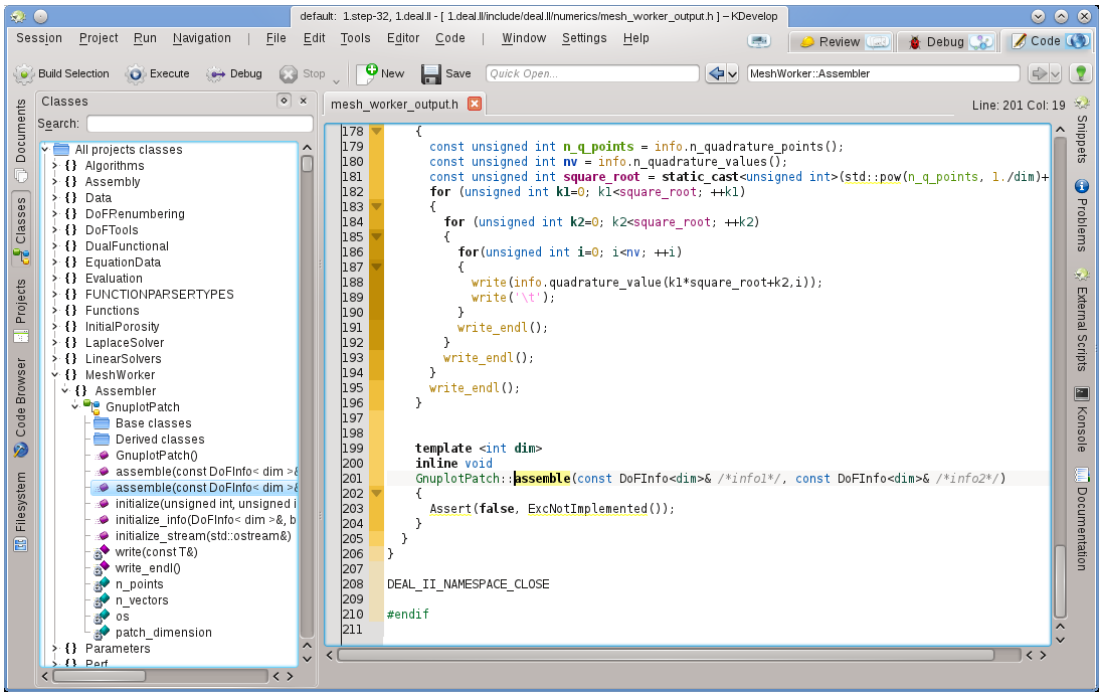
  - Selezionando **Navigazione** → **Salta alla definizione** (o premendo **Ctrl-,**).
  - Selezionando **Navigazione** → **Contesto visitato precedente** (o premendo **Meta-Sinistra**), come è stato descritto sopra.

NOTA
Saltare alla dichiarazione di un simbolo è qualcosa che non funziona solo quando si mette il cursore sul nome delle funzione che stai implementando al momento. In effetti, funziona anche per altri simboli: mettendo il cursore su una variabile (locale, globale o membro) e saltando alla sua dichiarazione si viene portati alla posizione della dichiarazione. Allo stesso modo puoi mettere il cursore sul nome della classe, per esempio su una variabile di una dichiarazione di funzione, e saltare nella posizione della sua dichiarazione.

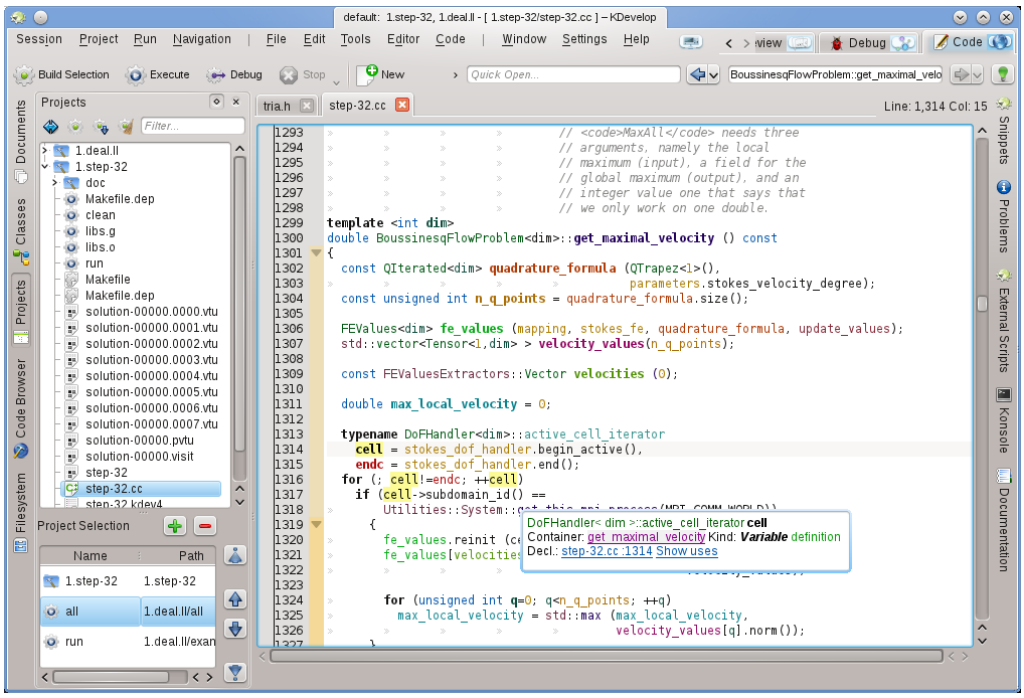
- **Commuta tra definizione/dichiarazione**: nell'esempio precedente, per saltare al punto in cui è definita la funzione corrente, dovevi mettere prima il cursore sul nome della funzione. Per evitare questo passo, puoi selezionare **Navigazione** → **Commuta definizione/dichiarazione** (o premere **Shift-Ctrl-C**) per saltare alla dichiarazione della funzione sul quale è presente il cursore. La selezione della stessa voce del menu una seconda volta ti riporterà nel posto in cui è definita la funzione.
- **Uso di precedente/successivo**: mettendo il cursore sul nome di una variabile locale e selezionando **Navigazione** → **Uso successivo** (o premendo **Meta-Shift-Destra**) si viene portati all'uso successivo di questa variabile nel codice. (Nota che questo non solo cerca l'occorrenza successiva del nome della variabile ma tiene conto del fatto che le variabili con lo stesso nome ma in differenti ambiti sono diverse). La stessa cosa vale per l'uso dei nomi delle funzioni. Selezionando **Navigazione** → **Uso precedente** (o premendo **Meta-Shift-Sinistra**) si viene portati all'uso precedente di un simbolo.

NOTA
Per visualizzare un elenco di tutti gli usi di un nome, posiziona il cursore su di esso e apri lo strumento **Browser del codice** o premi e tieni premuto il pulsante **Alt**. Questo è spiegato in dettaglio nella sezione Esplorare il codice.

- L'**elenco dei contesti**: i browser web hanno questa caratteristica con la quale si può andare indietro o avanti nell'elenco delle pagine visitate più di recente. KDevelop ha lo stesso tipo di caratteristica, a parte il fatto che invece di pagine web si visitano *contesti*. Un contesto è la posizione corrente del cursore, e puoi cambiarla allontanandoti da questa usando qualsiasi cosa eccetto i cursori — per esempio, facendo clic su una posizione fornita da un suggerimento nella vista strumento **Browser del codice**, una delle opzioni date nel menu **Navigazione**, o qualsiasi altro comando. Usando **Navigazione** → **Contesto visitato precedente** (**Meta-Sinistra**) e **Navigazione** → **Contesto visitato successivo** (**Meta-Destra**) si viene portati all'elenco dei contesti visitati proprio come i pulsanti **indietro** e **avanti** di un browser ti portano alla pagina precedente o prossima tra quelle che hai visitato.
- Infine, ci sono viste strumenti che ti permettono di navigare in posti diversi del tuo codice. Per esempio, lo strumento **Classi** ti fornisce un elenco di tutti i namespace e classi di tutti i progetti della sessione corrente, e ti permette di espanderli per vedere le funzioni e variabili membro di ognuna di queste classi:



Il doppio clic su un elemento (o scorrendo il menu contestuale usando il tasto destro del mouse) ti permette di saltare alla posizione della dichiarazione dell'elemento. Gli altri strumenti ti permettono di fare cose simili; per esempio, la vista strumento **Progetti** fornisce un elenco di file che sono parte di una sessione:



Un ulteriore doppio clic su un file lo apre.

3.4 Scrivere codice sorgente

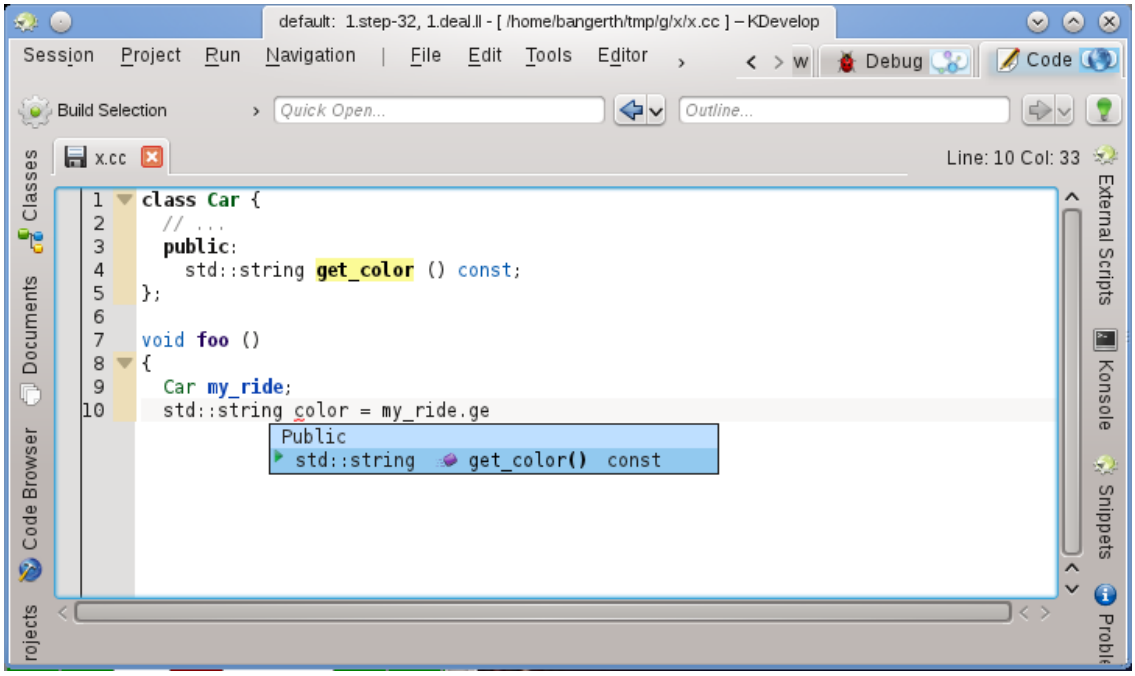
Poiché KDevelop interpreta il codice sorgente dei tuoi progetti, può essere d'aiuto nello scrivere più codice. Nel seguito vengono presentati alcuni dei modi nel quale questo può essere fatto.

3.4.1 Auto-completamento

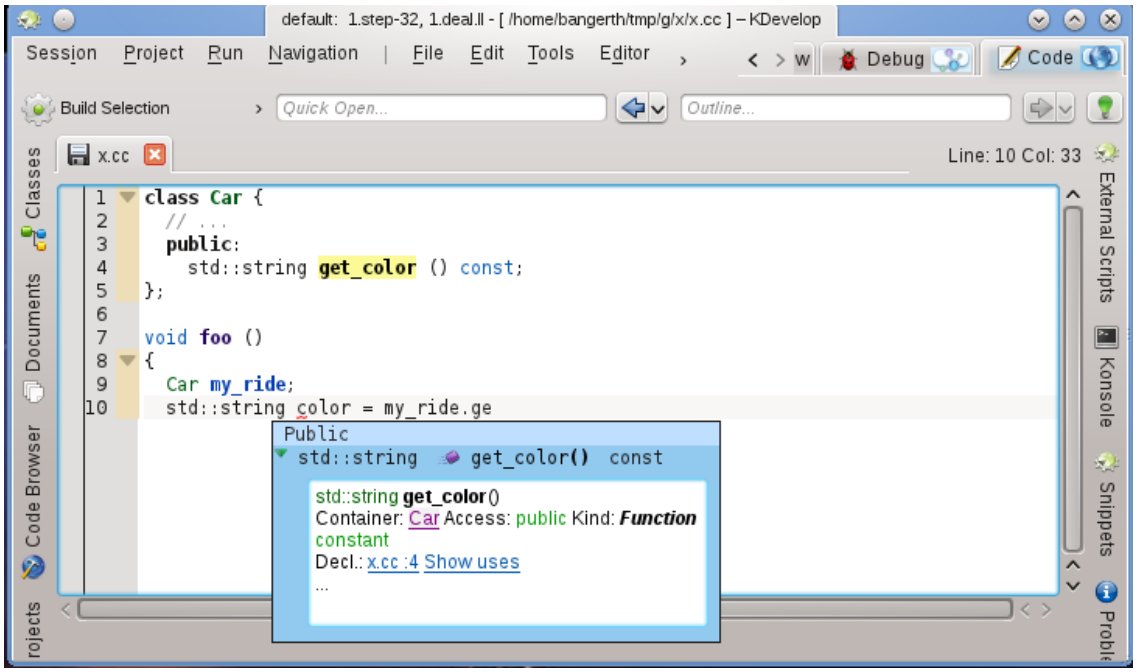
Probabilmente la caratteristica più utile di tutte nello scrivere nuovo codice è l'auto-completamento. Considera, per esempio, il seguente pezzo di codice:

```
class Car {
  // ...
  public:
    std::string get_color () const;
};
void foo()
{
  Car my_ride;
  // ...do something with this variable...
  std::string color = my_ride.ge
```

Nell'ultima riga, KDevelop ricorderà che la variabile `my_ride` è di tipo `Car`, e automaticamente proporrà di completare il nome della funzione membro `ge` come `get_color`. In realtà, tutto quello che devi fare è continuare a scrivere finché la funzione di auto-completamento ha ridotto le corrispondenze ad una, e poi premere il tasto **Invio**:



Nota che puoi fare clic sul suggerimento per avere più informazioni sulla funzione a parte il suo tipo di ritorno e se è pubblica:



L'auto-completamento può salvarti dal digitare molti caratteri se il tuo progetto usa nomi lunghi per variabili e funzioni; inoltre, evita errori di ortografia (e i risultanti errori di compilazione) e rende più semplici da ricordare i nomi esatti delle funzioni; per esempio, se tutti i vostri getter iniziano con `get_`, allora la funzione di auto-completamento sarà capace solo di presentarti un elenco di getter possibili quando hai digitato le prime quattro lettere, probabilmente ricordandoti nel processo quale delle funzioni è quella giusta. Nota che affinché l'auto-completamento funzioni, né la dichiarazione della classe `Car` né quella della variabile `my_ride` devono essere nello stesso file in cui si stà scrivendo il codice. KDevelop deve semplicemente sapere che queste classi e variabili sono connesse, cioè i file nei quali queste connessioni sono fatte devono essere parte del progetto sul quale stai attualmente lavorando.

NOTA

KDevelop non sa sempre quando esserti d'aiuto nel completare il codice. Se il suggerimento dell'auto-completamento non si apre automaticamente, premi **Ctrl-Spazio** per aprire un elenco da cui sceglierne uno. In generale, affinché l'auto-completamento funzioni, KDevelop deve analizzare i tuoi file sorgente. Questo viene fatto in background per tutti i file che fanno parte dei progetti della sessione corrente dopo che hai avviato KDevelop, subito dopo aver smesso di scrivere (il ritardo può essere configurato).

NOTA

KDevelop analizza soltanto i file che considera codice sorgente, determinandolo dal tipo MIME del file. Questo tipo è stabilito la prima volta che il file viene salvato; di conseguenza, creare un nuovo file e iniziare a scriverci codice non avvierà l'analisi dell'auto-completamento fino a quando non lo si salva per la prima volta.

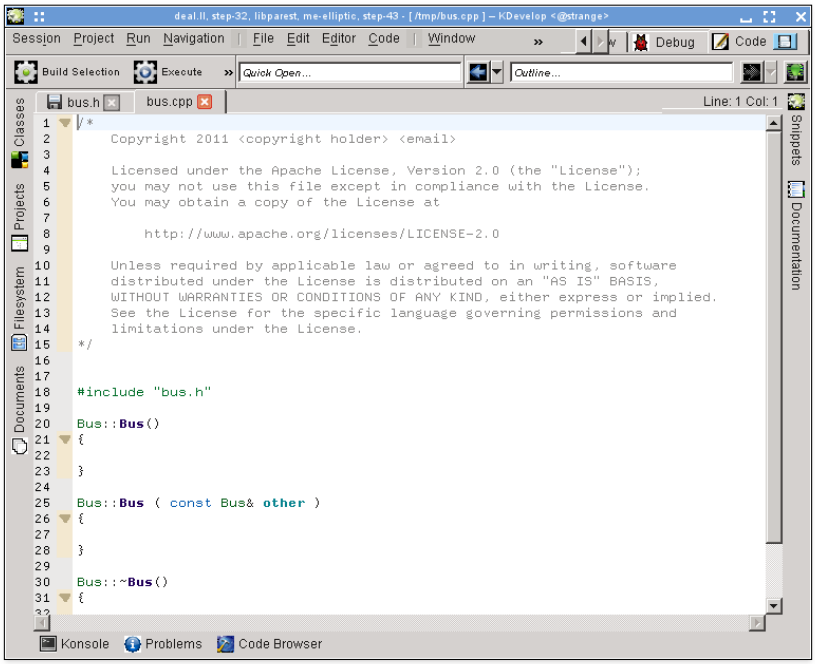
NOTA
Come nella nota precedente, affinché l'auto-completamento funzioni, KDevelop deve essere in grado di trovare le dichiarazioni nei file di intestazione. Per questo cerca in un numero di percorsi predefiniti. Se non trova automaticamente un file di intestazione, sottolinea il nome del file di intestazione in rosso; in questo caso, fai clic col tasto destro sul nome sottolineato per dire esplicitamente a KDevelop dove trovare questi file e le informazioni che questi forniscono.

NOTA
La configurazione dell'auto-completamento è discussa in questa sezione del manuale.

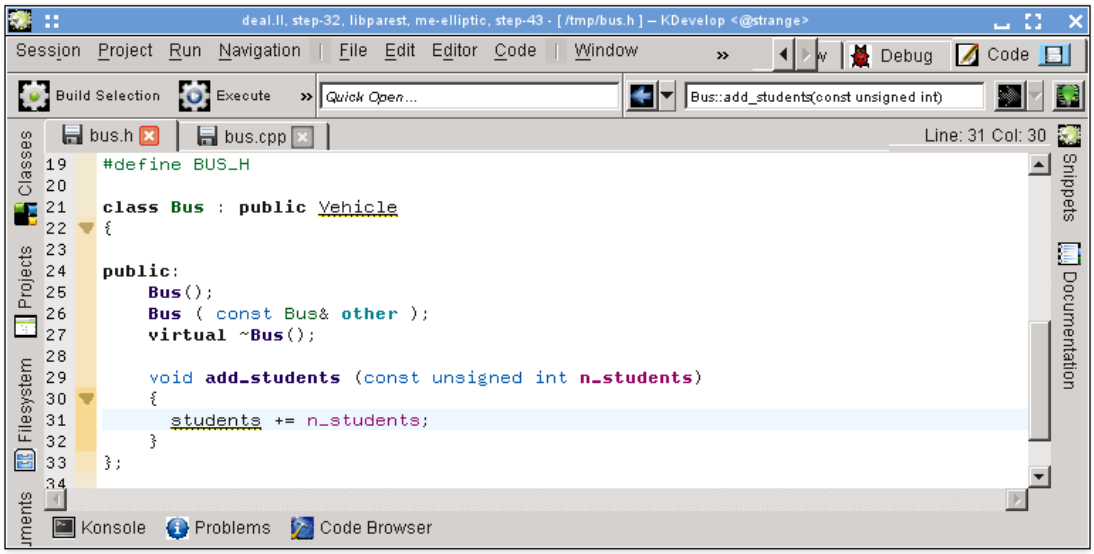
3.4.2 Aggiungere nuove classi e implementare le funzioni membro

KDevelop ha un assistente per aggiungere nuove classi. La procedura è descritta in Creare una nuova classe. Si può creare una semplice classe in C++ scegliendo il modello Base C++ dalla categoria `Classe`. Nell'assistente, si possono scegliere alcune funzioni membro predefinite, per esempio un costruttore vuoto, un costruttore di copie e un distruttore.

Dopo che l'assistente ha completato, i nuovi file vengono creati e aperti nell'editor. Il file di intestazione include già le include guard e le nuove classi hanno tutte le funzioni membro scelte. I successivi due passi sarebbero per documentare la classe e le sue funzioni membro e per la loro implementazione. Si discuterà degli aiuti per la documentazione delle classi e delle funzioni qui sotto. Per implementare le funzioni speciali già aggiunte, andare semplicemente nella scheda **bus.cpp** dove sono già forniti gli scheletri delle funzioni:

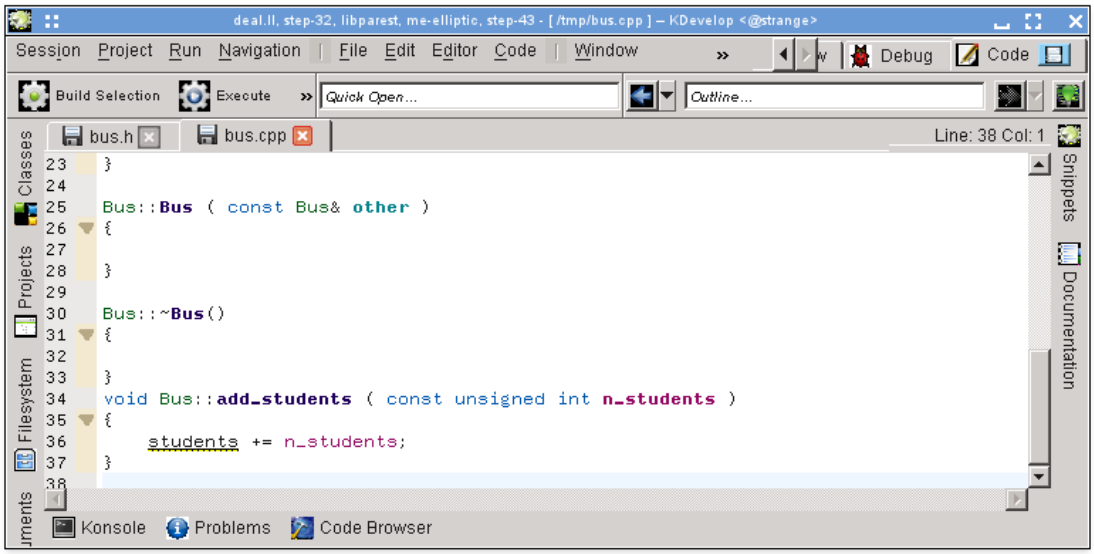


Per aggiungere nuove funzioni membro, ritornare nella scheda **bus.h** e aggiungere il nome di una funzione. Per esempio, aggiungiamo questa:



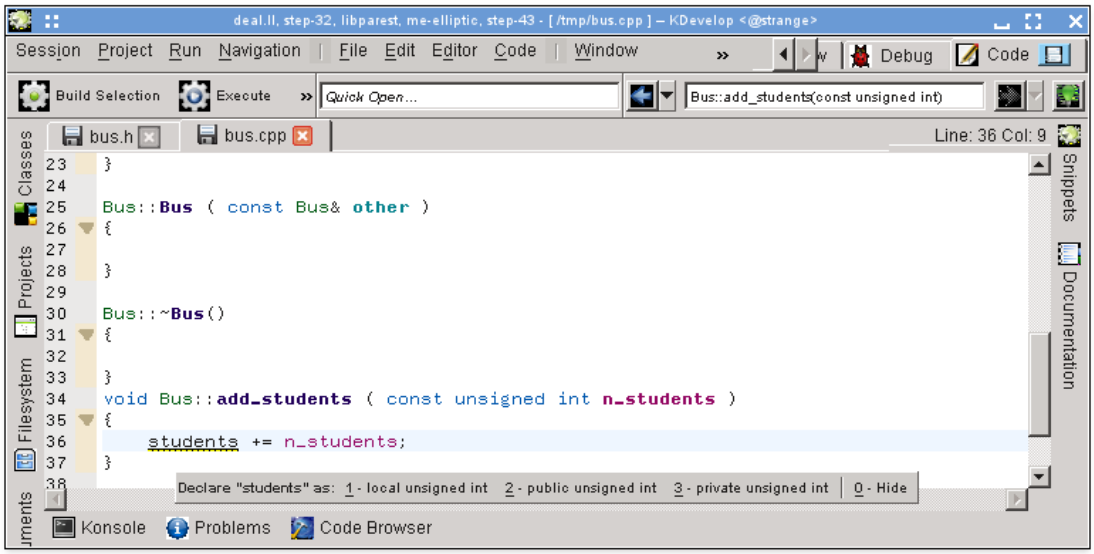

```
19  #define BUS_H
20
21  class Bus : public Vehicle
22  {
23
24  public:
25      Bus();
26      Bus ( const Bus& other );
27      virtual ~Bus();
28
29      void add_students (const unsigned int n_students)
30      {
31        students += n_students;
32      }
33  };
34
```

Nota come ho iniziato l'implementazione. Comunque, in molti stili di codifica, la funzione non dovrebbe essere implementata nel file di intestazione ma piuttosto nel corrispondente file .cpp. A tal fine, posizionare il cursore sul nome della funzione e selezionare **Codice** → **Sposta il sorgente** o premere **Ctrl-Alt-S**. Questo rimuoverà il codice tra le parentesi graffe dal file di instestazione (e lo sostituirà con un punto e virgola allo scopo di terminare la dichiarazione della funzione) e lo sposterà nel codice sorgente:

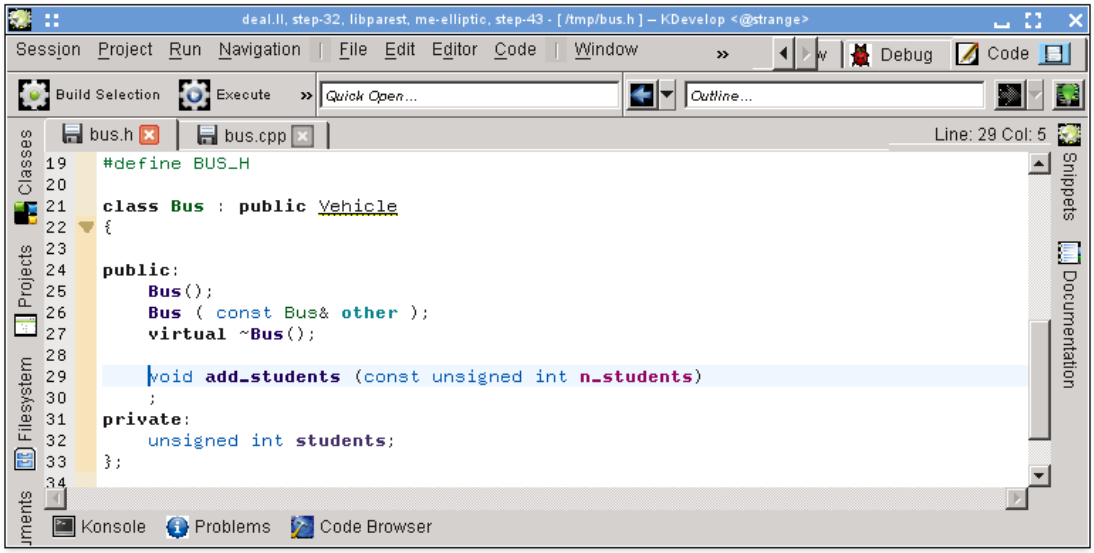



```
23  }
24
25  Bus::Bus ( const Bus& other )
26  {
27
28  }
29
30  Bus::~Bus()
31  {
32
33  }
34  void Bus::add_students ( const unsigned int n_students )
35  {
36      students += n_students;
37  }
38
```

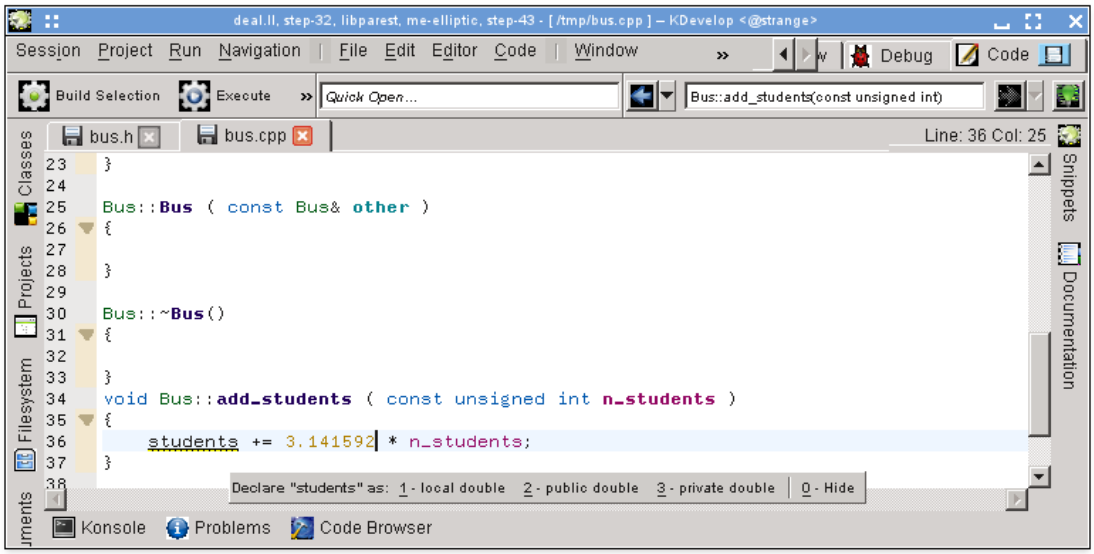
Nota che quello che volevo sottointendere è che la variabile `students` dovrebbe essere probabilmente una variabile membro della classe `Bus` ma che non ho ancora aggiunto. Nota pure come KDevelop la sottolinea per rendere chiaro che non sa niente della variabile. Ma questo problema può essere risolto: facendo clic sul nome della variabile si ottiene il seguente suggerimento:



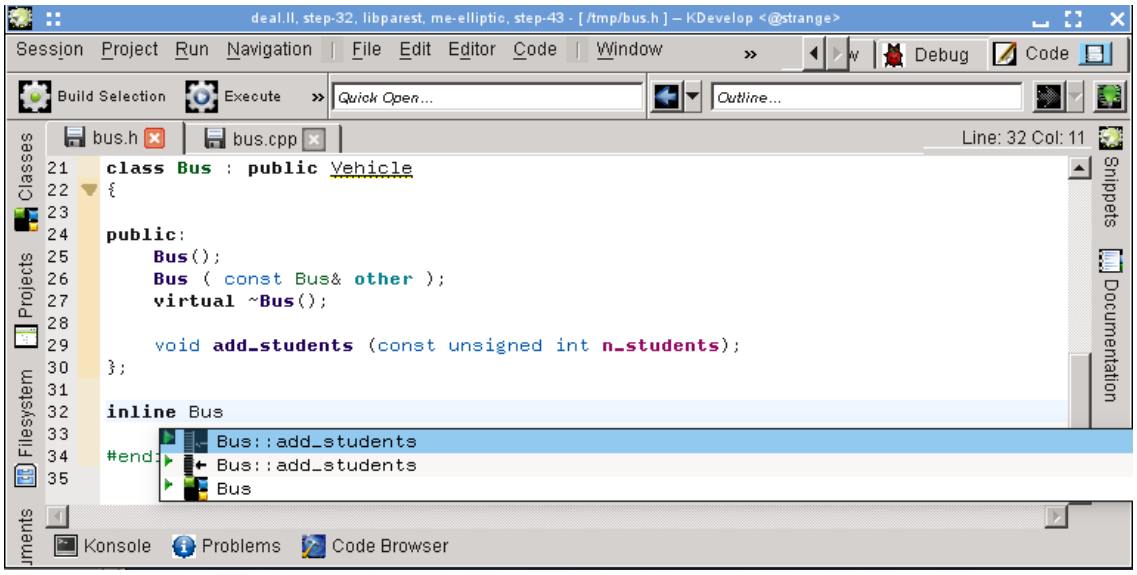
(Lo stesso risultato può essere raggiunto facendo clic con il tasto destro su di esso e selezionando **Risolvere: Dichiara come**.) Ora consideriamo '3 - private unsigned int' (o con il mouse, o premendo **Alt-3**) e vediamo come viene fuori nel file di intestazione:



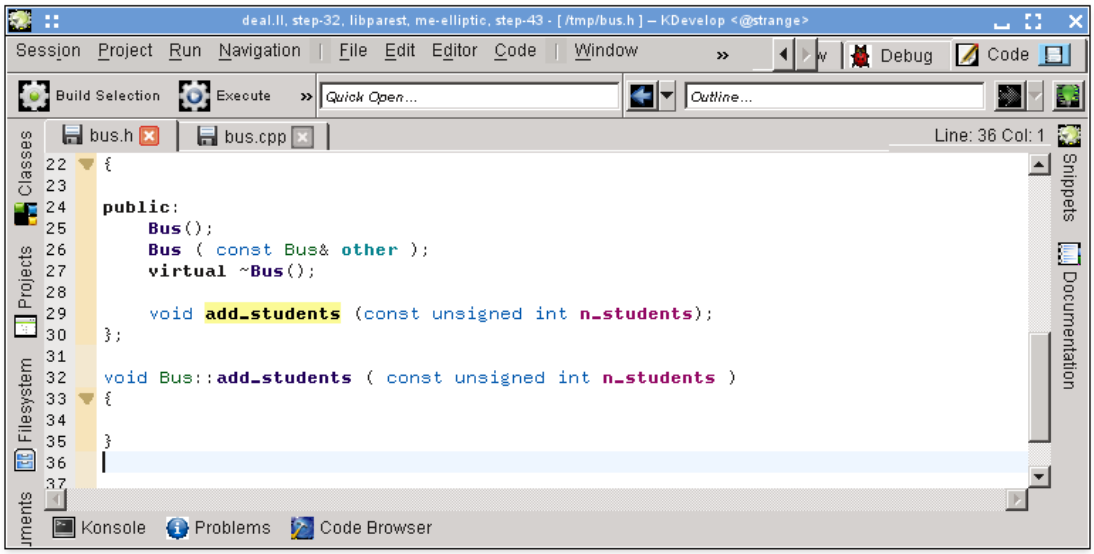
Vale la pena notare che KDevelop estrae il tipo della variabile che deve essere dichiarata dall'espressione usata per inizializzarla. Per esempio, se avessimo scritto la somma seguente in un modo piuttosto dubbio, avrebbe suggerito di dichiarare la variabile di tipo `double`:



Per concludere: il metodo che consiste nell'usare **Codice** → **Sposta il sorgente** non sempre inserisce la nuova funzione membro dove potresti volere. Per esempio, potresti volerla marcare come `inline` e metterla in fondo al file di intestazione. In un caso come questo, scrivi la dichiarazione e inizia a scrivere la definizione della funzione in questo modo:



KDevelop automaticamente offre tutti i possibili completamenti del caso. La selezione di una delle due voci `add_students` produce il codice seguente che riempie già l'elenco completo degli argomenti:




```
{

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

    void add_students (const unsigned int n_students);
};

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

}
```

NOTA

Nell'esempio, accettando una delle scelte che lo strumento di auto-completamento offre viene prodotta la firma completa ma sfortunatamente cancella l'indicazione `inline` scritta in precedenza. Questo comportamento è stato riportato nel bug 274245 di KDevelop.

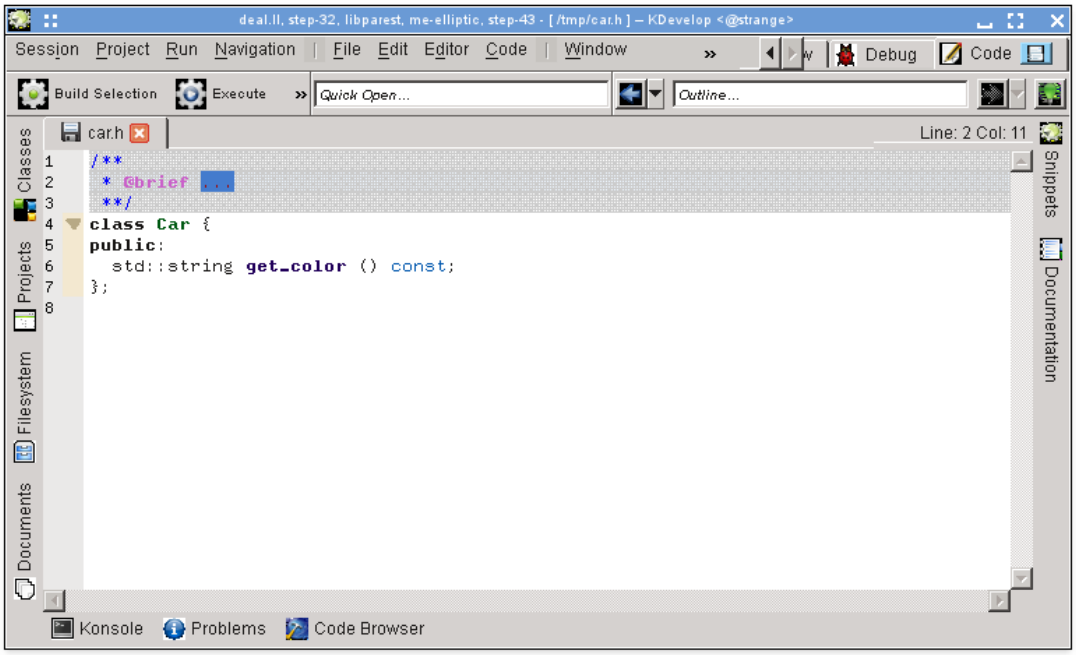
3.4.3 Documentare le dichiarazioni

Il buon codice è ben documentato, sia a livello dell'implementazione degli algoritmi delle funzioni che a livello dell'interfaccia — cioè, le classi, le funzioni (membro e locali) e le variabili devono essere documentate per spiegare a cosa servono, i possibili valori degli argomenti, pre e post condizioni, ecc.. Per quanto riguarda l'interfaccia della documentazione, doxygen è diventato lo standard de facto per la formattazione dei commenti che poi possono essere estratti e visualizzati da pagine web ricercabili.

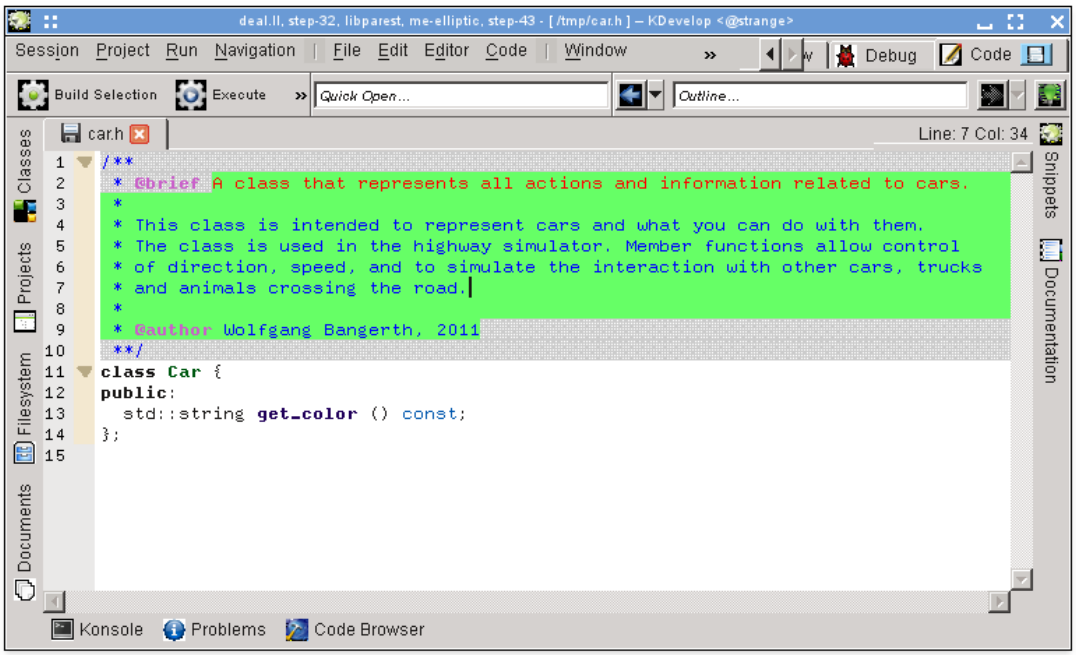
KDevelop supporta questo stile di commenti fornendo una scorciatoia per generare l'infrastruttura dei commenti che documenta una classe o una funzione membro. Per esempio, immagina di aver già scritto questo codice:

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

Ora vuoi aggiungere la documentazione sia alla classe che alla funzione membro. A tal fine, sposta il cursore sulla prima riga e seleziona **Codice** → **Documenta dichiarazione** o premi **Alt-Shift-D**. KDevelop risponderà in questo modo:



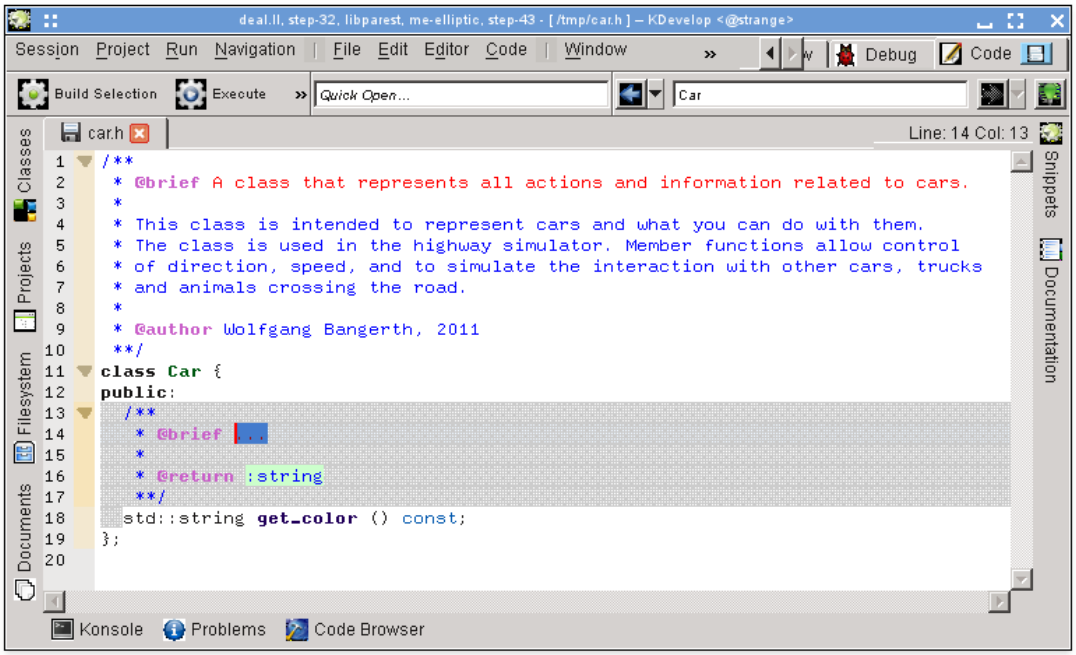
Il cursore si trova già nell'area in grigio in modo che tu possa scrivere una breve descrizione (dopo la parola chiave di doxygen `@brief`) di questa classe. Poi puoi continuare ad aggiungere documentazione a questo commento in modo da avere una panoramica più dettagliata di quello che fa la classe:



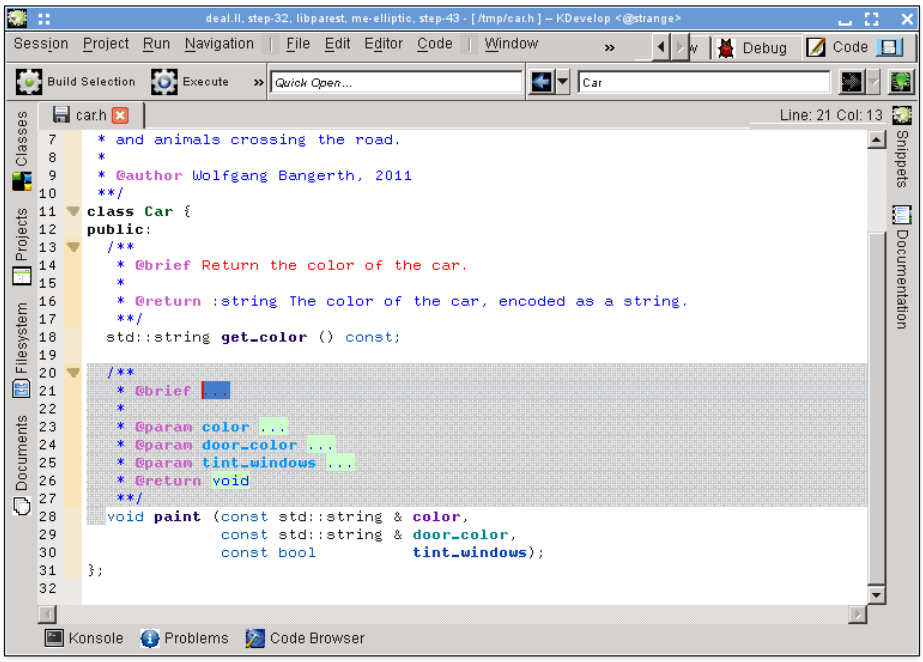
Mentre l'editor si trova all'interno del commento, il testo del commento è evidenziato in verde (l'evidenziazione scompare quando sposti il cursore al di fuori del commento). Quando raggiun-

gi la fine di una riga, premi **Invio** e KDevelop inizierà automaticamente una nuova riga che inizia con un asterisco e posizionerà il cursore di un carattere rientrato.

Ora documentiamo la funzione membro, metti di nuovo il cursore sulla riga della dichiarazione e seleziona **Codice** → **Documenta dichiarazione** o premi **Alt-Shift-D**:



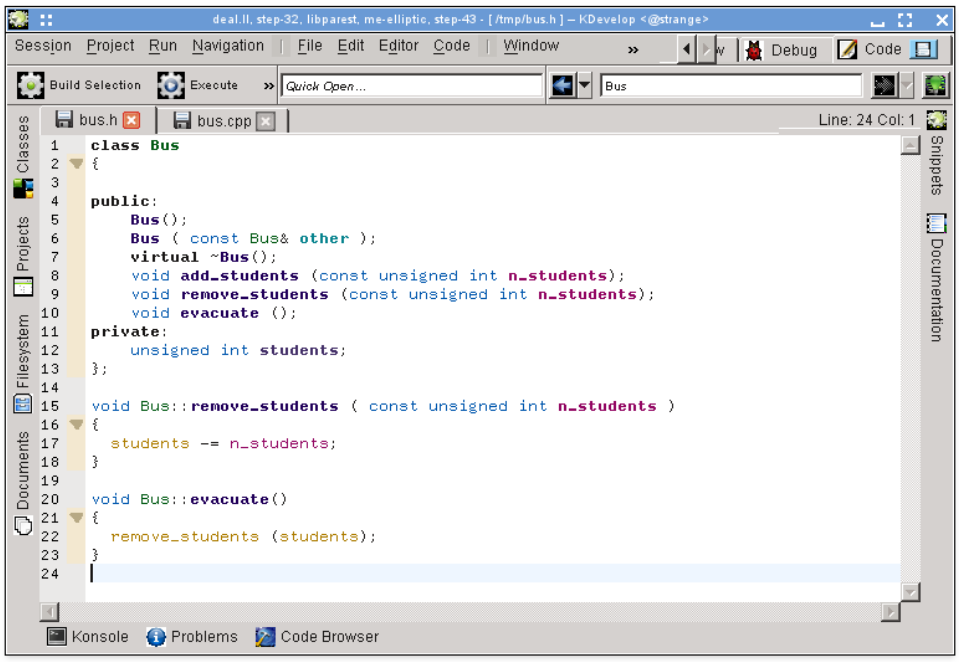
Di nuovo, KDevelop genererà automaticamente lo scheletro di un commento, insieme alla documentazione della stessa funzione, in aggiunta al suo tipo di ritorno. In questo caso, il nome della funzione si spiega da sola, ma spesso gli argomenti della funzione potrebbero non essere documentati individualmente anche se dovrebbero. Per illustrare quanto appena detto, consideriamo una funzione un po' più interessante e il commento che KDevelop genera automaticamente:



Qui, il commento proposto contiene già tutti i campi per i parametri individuali di Doxygen, per esempio.

3.4.4 Rinominare le variabili, le funzioni e le classi

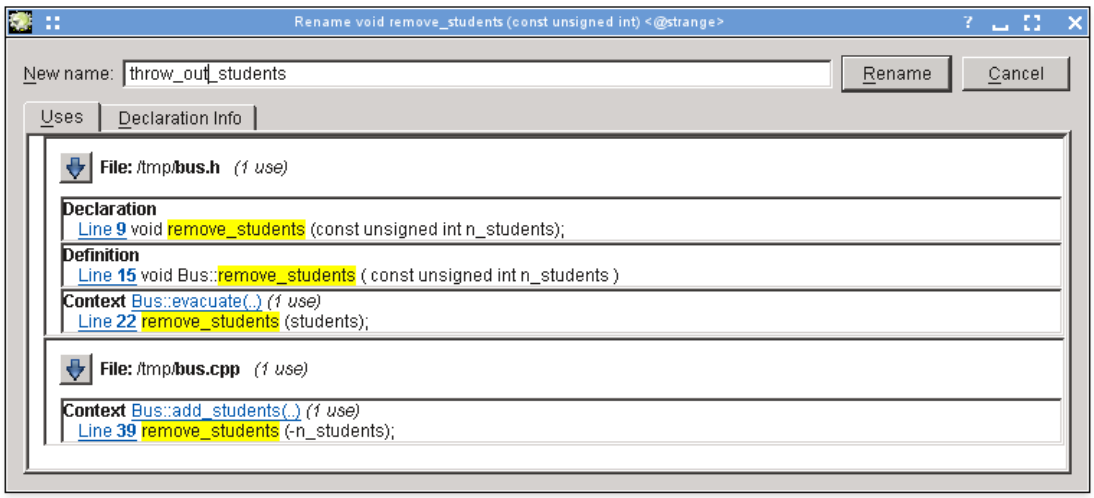
A volte, si vuole rinominare una funzione, una classe o una variabile. Per esempio, diciamo di avere già questo:



Poi ci rendiamo conto di non essere contenti del nome `remove_students` e preferiremo chiamarlo, ad esempio, `throw_out_students`. Potremmo fare cerca e sostituisci il nome, ma questo ha due inconvenienti:

- La funzione potrebbe essere usata in più di un file.
- In realtà vogliamo solo rinominare questa funzione e non toccare le funzioni che potrebbero avere lo stesso nome ma che sono dichiarate in altre classi o namespace.

Antrambi questi problemi possono essere risolti spostando il cursore su una delle occorrenze del nome della funzione e selezionando **Codice** → **Rinomina dichiarazione** (o facendo clic con il tasto destro sul nome e selezionando **Rinomina Bus::remove_students**). Questo fa apparire una finestra di dialogo in cui è possibile inserire il nuovo nome della funzione e dove si possono anche vedere tutti i posti in cui la funzione è effettivamente usata:

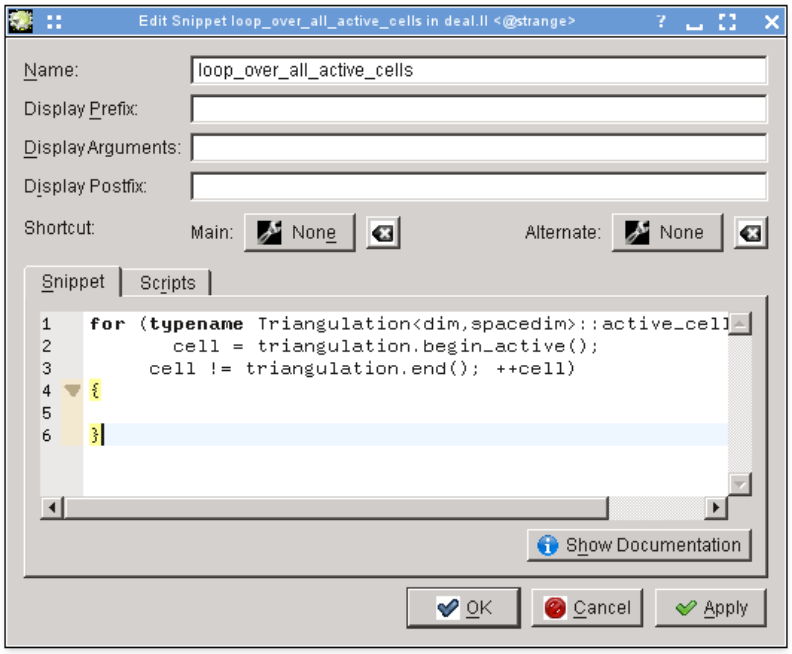


3.4.5 Frammenti di codice

La maggior parte dei progetti hanno parti di codice che frequentemente devono essere scritte nel codice sorgente. Esempi sono: chi scrive compilatori, un ciclo su tutte le istruzioni; chi realizza interfacce utente, controllare che l'input dell'utente sia valido e se non aprire una finestra di errore; nel progetto dell'autore di queste righe, sarebbe codice del tipo

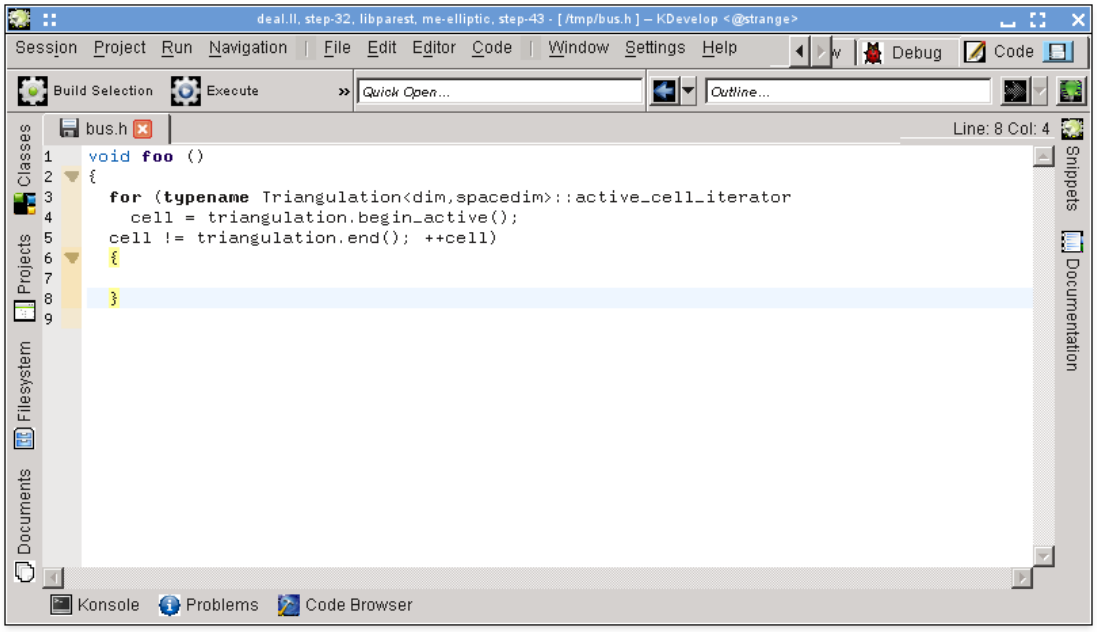
```
for (typename Triangulation::active_cell_iterator
       cell = triangulation.begin_active();
     cell != triangulation.end(); ++cell)
  ... do something with the cell ...
```

Piuttosto che scrivere continuamente del testo simile (con tutti gli errori di battitura connessi che si introducono), qui può essere d'aiuto lo strumento **Frammenti** di KDevelop. A tale fine, aprire la vista strumento (vedi Strumenti e viste se il tasto corrispondente non è già nella finestra). Poi fare clic sul tasto 'Aggiungi un deposito' (un nome improprio — ti consente di creare una raccolta di frammenti di codice sorgente di un tipo particolare con un nome, ad es. Sorgenti C++) e creare un deposito vuoto. Poi fare clic su  per aggiungere un frammento, ottenendo una finestra di dialogo come la seguente:



> NOTA
> Il nome del frammento potrebbe non vere spazi o altri caratteri speciali perché deve sembrare un normale nome di funzione o variabile (per ragioni che saranno chiare nel paragrafo successivo).

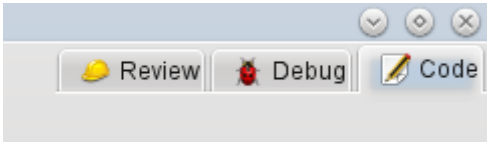
Per usare un frammento definito in questo modo, quando stai editando il codice, è sufficente scrivere il nome del frammento nello stesso modo in cui avresti scritto il nome di qualsiasi altra funzione o variabile. Questo nome diventerà disponibile all'auto-completamento — ciò significa che non c'è niente di male nell'uso di nomi lunghi e descrittivi per i frammenti come quello qui sopra — e quando accetti il suggerimento dell'auto-completamento (per esempio premento **Invio**), la parte del nome del frammento già inserito sarà rimpiazzata dal nome completo del frammento e sarà adeguatamente identata:



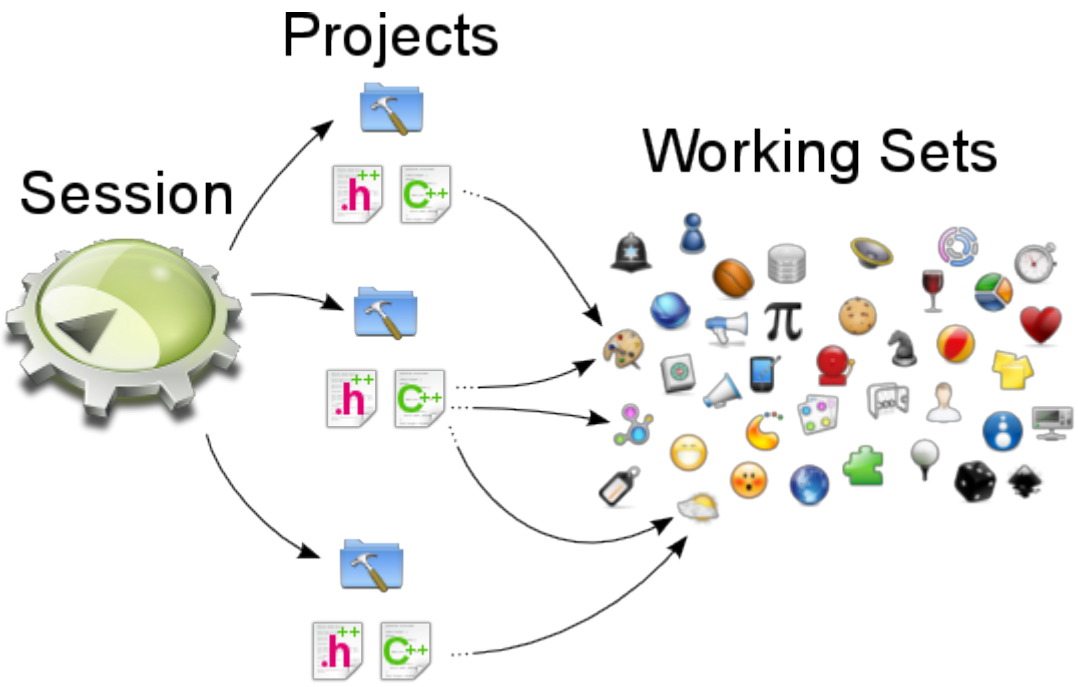
Nota che affinché questo funzioni, deve essere aperta e visibile la vista strumento **Frammenti**: hai bisogno unicamente e solamente della vista strumento per definire dei nuovi frammenti. Un modo alternativo, anche se meno conveniente, di espandere un frammento consiste nel fare semplicemente clic sullo stesso nella relativa vista strumento.

> NOTA
> I frammenti sono molto più potenti di quanto appena spiegato. Per una completa descrizione di cosa puoi farci, vedi la documentazione dettagliata dello strumento frammenti.

3.5 Modalità e working set

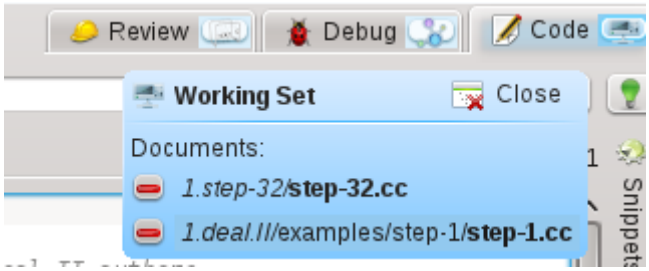


Se sei arrivato fino a questo punto, dai un'occhiata in alto a destra della finestra principale di KDevelop: come mostrato in figura, vedrai che vi sono tre **modalità** in cui si può trovare KDevelop: **Codice** (la modalità di cui abbiamo discusso in questo capitolo), **Debug** (vedi Fare il debug dei programmi) e **Revisione** (vedi Lavorare con i sistemi di controllo versione).



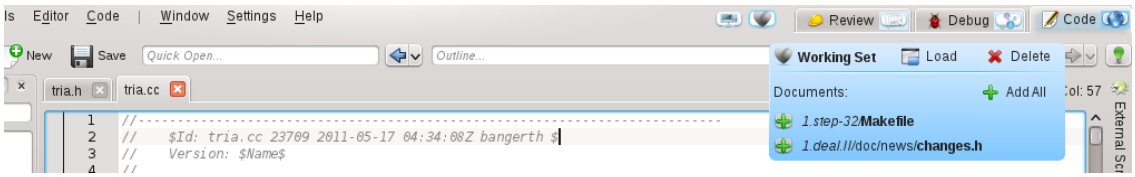
Ogni modalità ha il proprio insieme di strumenti disposti sulla finestra della stessa, e ogni modalità ha anche un *working set* di documenti e file attualmente aperti. Inoltre, ogni working set è associato alla sessione corrente, cioè abbiamo la relazione mostrata sopra. Nota che i file nel working set provengono dalla stessa sessione, ma potrebbero venire da progetti diversi che fanno parte della stessa sessione.

Se apri per la prima volta KDevelop, il working set è vuoto — non ci sono file aperti. Ma quando apri i file per la modifica (o per il debug, o per la revisione nelle altre modalità) il tuo working set diventa più grande. Il fatto che il tuo working set non è vuoto è indicato da un simbolo nella scheda, come mostrato qui sotto. Noterai che ogni volta che chiudi KDevelop e poi lo riavvii, il working set viene salvato e ripristinato, cioè ottieni lo stesso insieme di file aperti.



Se passi il mouse sul simbolo del working set, ottieni un suggerimento che ti mostra quali file sono aperti in questo working set (qui: i file `step-32.cc` and `step-1.cc`). Facendo clic sul segno meno di color rosso si chiude la scheda del file corrispondente. Forse ancora più importante, facendo clic sul pulsante con il nome corrispondente ti permette di **chiudere** tutto il working set in una volta (cioè chiudere tutti i file aperti). Il punto sulla chiusura di un working set, comunque, è che non chiude soltanto tutti i file, in realtà salva il working set e ne apre uno nuovo, ancora vuoto. Lo puoi vedere qui:

Nota i due simboli a sinistra delle tre schede delle modalità (il cuore e il simbolo non identificabile alla sua sinistra). Ognuno di questi due simboli rappresenta un working set salvato, in aggiunta al working set aperto. Se passi il mouse sopra al cuore, avrai qualcosa del genere:



Ti mostra che il working set corrispondente contiene due file con i rispettivi nomi: `Makefile` e `changes.h`. Facendo clic su **Carica** il working set corrente verrà chiuso e salvato (che come mostrato qui ha i seguenti file aperti `tria.h` e `tria.cc`) al posto di aprire il working set selezionato. Puoi anche cancellare per sempre un working set, questo lo rimuove dall'insieme dei working set salvati.

3.6 Alcune utili scorciatoie da tastiera

L'editor di KDevelop segue lo standard delle scorciatoie da tastiera per le operazioni di modifica usuali. Comunque, supporta anche un numero maggiore di operazioni avanzate quando si modifica il codice sorgente, alcune delle quali sono legate a particolari combinazioni di tasti. Le seguenti sono particolarmente utili:

Saltare da un posto a un altro nel codice	
Ctrl-Alt-O	Apertura veloce file: inserire parte del nome di un file e selezionare tra tutti i file nell'albero delle cartelle di progetto della sessione corrente che corrisponde con la stringa; il file poi verrà aperto
Ctrl-Alt-C	Apertura veloce classe: inserire parte del nome di una classe e selezionare tra tutti i nomi quello corrispondente; il cursore poi salterà alla dichiarazione della classe
Ctrl-Alt-M	Apertura veloce funzione: inserire parte del nome di una funzione (membro) e selezionare tra tutti i nomi quello corrispondente; nota che l'elenco mostra sia le dichiarazioni che le definizioni e il cursore salterà alla voce selezionata
Ctrl-Alt-Q	Apertura veloce universale: digita qualsiasi cosa (nome di un file, nome di una classe, nome di una funzione) e otterai un'elenco di tutto quello che corrisponde
Ctrl-Alt-N	Schema riassuntivo: fornisce un elenco di tutte le cose avvenute a questo file, ad es. dichiarazioni di classi e definizioni di funzioni
Ctrl-,	Salta alla definizione di una funzione se il cursore è sulla dichiarazione di una funzione

Ctrl-.	Salta alla dichiarazione di una funzione o variabile se il cursore è nella definizione di una funzione
Ctrl-Alt-Pag↓	Funzione successiva
Ctrl-Alt-Pag↑	Funzione precedente
Ctrl-G	Vai alla riga

Cercare e sostituire	
Ctrl-F	Trova
F3	Trova successivo
Ctrl-R	Sostituisci
Ctrl-Alt-F	Cerca-Sostituisci in file multipli

Altre cose	
Ctrl-_	Riduce un livello: rimuovi questo blocco dalla vista, per esempio se vuoi concentrarti sulla figura più grande all'interno di una funzione
Ctrl-+	Espande un livello: annulla la riduzione
Ctrl-D	Commenta il testo selezionato o la riga corrente
Ctrl-Shift-D	Decommenta il testo selezionato o la riga corrente
Alt-Shift-D	Documenta la funzione corrente. Se il cursore è sulla dichiarazione di una classe o funzione allora premere questa combinazione di tasti creerà un commento in stile doxygen pre-popolato con una lista di tutti i parametri, valori di ritorno, ecc.
Ctrl-T	Inverte il carattere corrente e quello precedente
Ctrl-K	Cancella la riga corrente (nota: non è soltanti il 'cancella da qui alla fine della riga' di emacs)

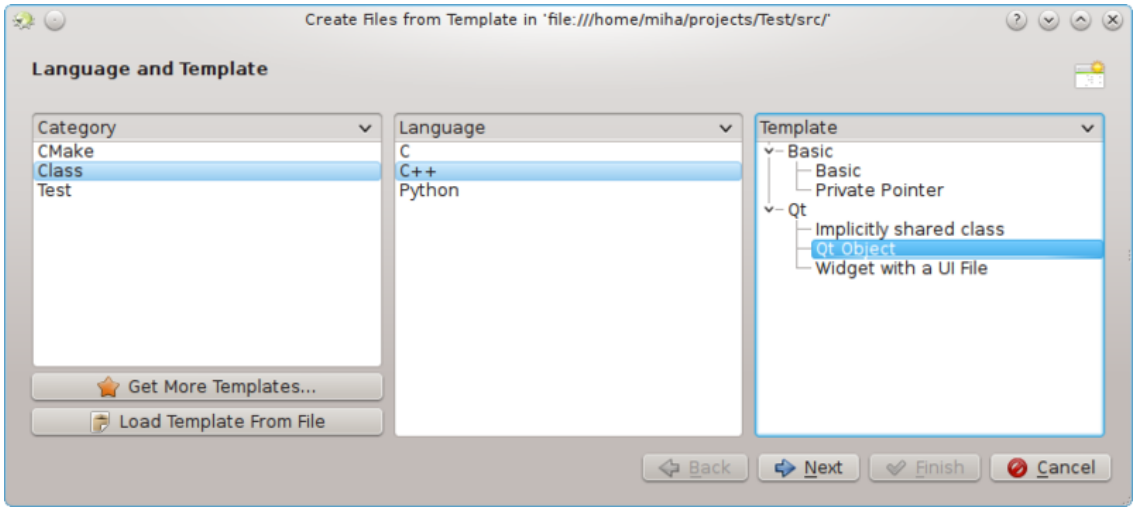
Capitolo 4

Generare codice con i modelli

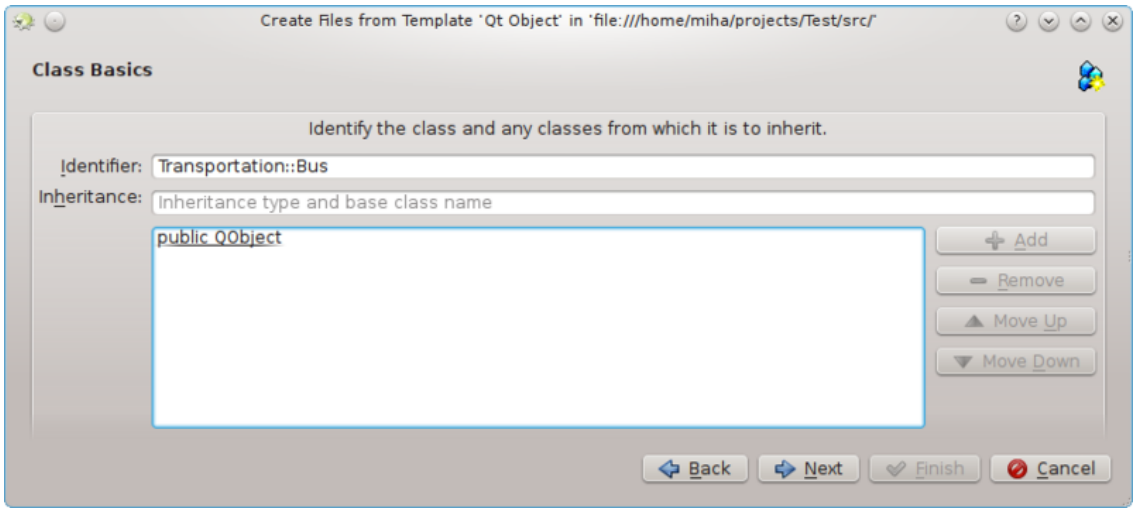
KDevelop usa i modelli per generare file di codice sorgente e per evitare di scrivere codice ripetibile.

4.1 Creare una nuova classe

Il modo più banale di generare codice probabilmente è scrivere nuove classi. Per creare una nuova classe in un progetto esistente, fare clic con il tasto destro sulla cartella di un progetto e scegliere **Crea da modello...**. Si può fare la stessa cosa dal menu facendo clic su **File** → **Nuovo da modello...**, ma usare la cartella del progetto ha il vantaggio di impostare un collegamento base per i file di output. Scegli `Classe` nella vista di selezione della categoria, ed il linguaggio e il modello desiderati nelle altre due viste. Dopo aver selezionato il modello della classe, dovrai specificare i dettagli della nuova classe.

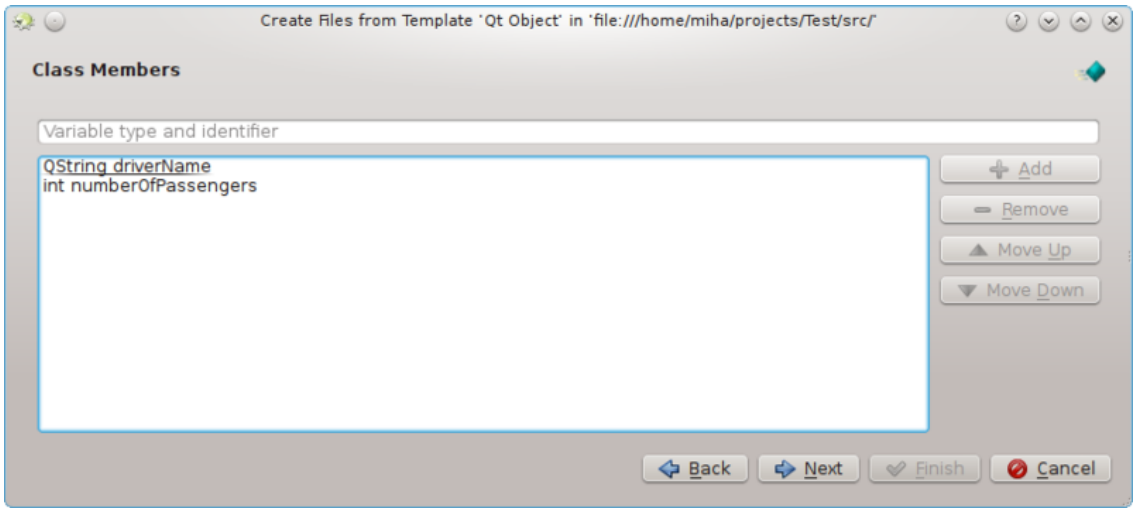


Come primo passo devi specificare un identificatore per la nuova classe. Questo può essere un nome semplice (come `Bus`) o un identificatore con un namespace (come `Transportation::Bus`). Nell'ultimo caso, KDevelop analizzerà l'identificatore e separerà correttamente il namespace dal nome effettivo. Nella stessa pagina, puoi aggiungere classi di base alla nuova classe. Puoi notare che alcuni modelli scelgono una classe base da soli, sei libero di rimuoverla e/o aggiungerne altre. Qui dovresti scrivere il tipo di ereditarietà, indipendentemente dal linguaggio, come ad esempio `public QObject` per il C++, `extends SomeClass` per il PHP o semplicemente il nome della classe per il Python.



Nella pagina seguente, ti verrà offerta una selezione di metodi virtuali dalle classi ereditate, così come alcuni costruttori, distruttori e operatori predefiniti. La selezione della casella di controllo accanto alla firma del metodo implementerà questo metodo nella nuova classe.

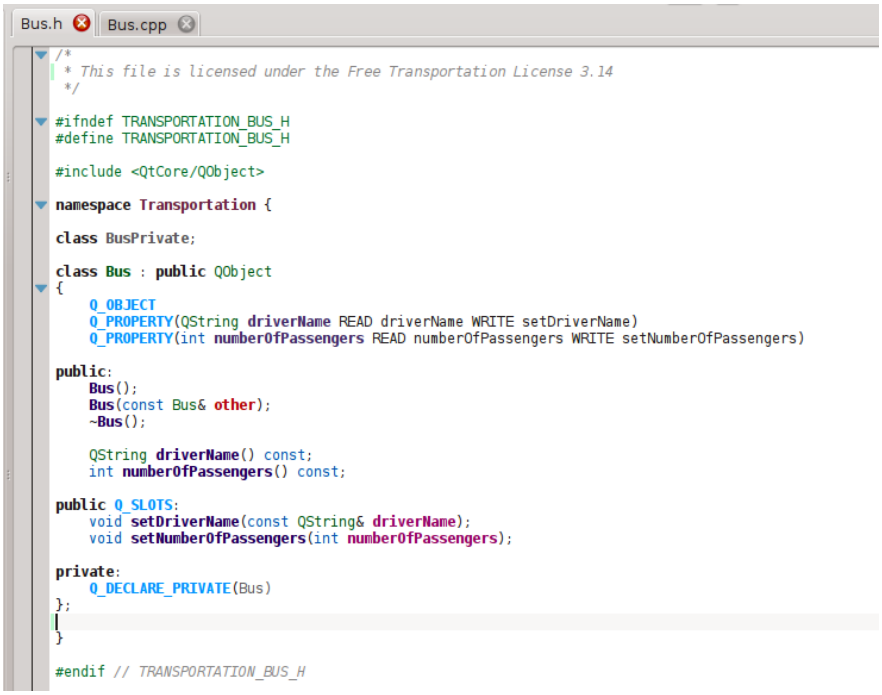
Facendo clic su **Successivo** si va ad una pagina dove puoi aggiungere membri ad una classe. A seconda del modello selezionato, questi possono apparire nella nuova classe come variabili membro, o il modello potrebbe creare per loro proprietà con setter e getter. In un linguaggio dove devono essere dichiarati i tipi di variabile, come il C++ ad esempio, devi specificare sia il tipo che il nome del membro, come `int number` o `QString name`. In altri linguaggi puoi non specificare il tipo, ma è buona norma specificarlo comunque, perché il modello scelto potrebbe ancora farne un certo uso.



Nelle pagine seguenti puoi scegliere una licenza per la tua nuova classe, impostare qualsiasi opzione personalizzata richiesta dal modello selezionato, e configurare dove salvare i file generati. Facendo clic su **Completa**, completi l'assistente e crei la nuova classe. I file generati saranno aperti nell'editor, in modo che tu possa iniziare ad aggiungere codice subito.

Dopo aver creato una nuova classe C++ ti sarà data la possibilità di aggiungere la classe al progetto scelto. Scegline uno dalle pagine della finestra, o chiudi la pagina e aggiungi manualmente i file ad un progetto.

Se hai scelto il modello `Qt Object`, controllato alcuni dei metodi predefiniti, e aggiunto due variabili membro, l'output dovrebbe essere simile alla seguente immagine.



Puoi vedere che i dati membro sono convertiti in proprietà Qt, con funzioni d'accesso e le macro Q_PROPERTY. Gli argomenti delle funzioni setter sono anche passati come const-reference, se del caso. Inoltre, è dichiarata una classe privata e un puntatore privato creato con Q_DECLARE_PRIVATE. Tutto questo è fatto dal modello, la scelta di un altro nel primo passo potrebbe cambiare completamente l'output.

4.2 Creare un nuovo unit test

Anche se la maggior parte delle infrastrutture per i test richiedono che ogni test sia anche una classe, KDevelop include un modo per semplificare la creazione di unit test. Per creare un nuovo test, fare clic col tasto destro sulla cartella di un progetto e scegliere **Crea da modello...**. Nella pagina di selezione dei modelli scegliere `Test` come categoria, poi scegliere il linguaggio di programmazione e il modello e fare clic su **Successivo**.

Ti saranno chiesti il nome del test e un'elenco dei casi di test. Per i casi di test, devi solo specificare un elenco di nomi. Alcune infrastrutture per unit test come PyUnit e PHPUnit, richiedono che i casi di test inizino con un prefisso speciale. In KDevelop, il modello è responsabile dell'aggiunta del prefisso, così non devi farlo tu. Dopo aver fatto clic su **Successivo**, indica la licenza e dove vengono generati i file, e il test verrà creato.

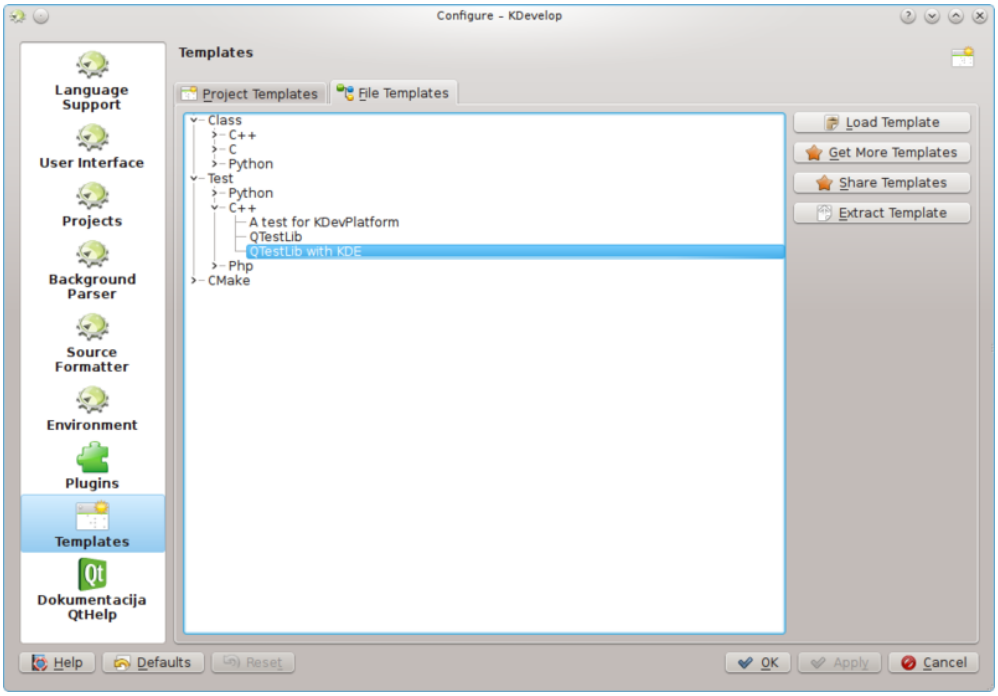
GlI unit test così creati non saranno aggiunti ad alcun obiettivo. Se stai usando CTest o qualche altra infrastruttura di test, accertati di aggiungere i nuovi file all'obiettivo.

4.3 Altri file

Mentre le classi e gli unit test ricevono attenzioni speciali durante la generazione di codice dai modelli, lo stesso metodo può essere usato da qualsiasi tipo di file di codice sorgente. Per esempio, si potrebbe usare un modello per un modulo Find di CMake o per un file. desktop. Questo può essere fatto scegliendo **Crea da modello...**, e selezionando la categoria e il modello che si vuole. Se la categoria selezionata non è né `Classe` né `Test`, avrai solo la possibilità di scegliere la licenza, le opzioni personalizzate specificate dal modello e dove salvare i file. Come per le classi e per i test, concluso l'assistente verranno generati i file che saranno aperti nell'editor.

4.4 Gestire i modelli

Dall'assistente **File** → **Nuovo da modello...**, puoi anche scaricare modelli aggiuntivi facendo clic sul pulsante **Ottieni altri modelli...**. Questo apre una finestra Scarica le Novità, da dove puoi installare modelli aggiuntivi, così come aggiornarli o rimuoverli. C'è anche un modulo di configurazione per i modelli, che può essere raggiunto facendo clic su **Impostazioni** → **Configura KDevelop** → **Modelli**. Da lì, puoi gestire sia i modelli dei file (spiegato qui sopra) che i modelli di progetto (utilizzati per la creare nuovi progetti).



Naturalmente, se nessuno dei modelli disponibili è adatto al tuo progetto, puoi sempre crearne di nuovi. Il modo più semplice probabilmente consiste nel copiare e modificare un modello esistente, un breve tutorial e un più lungo documento dettagliato ti saranno d'aiuto. Per copiare un modello installato, aprire il gestore di modelli facendo clic sul pulsante **Impostazioni** → **Configura KDevelop...** → **Modelli**, seleziona il modello che vuoi copiare, poi fai clic sul pulsante **Estrai modello**. Seleziona la cartella di destinazione, poi fai clic su **OK**, e il contenuto del modello verrà estratto nella cartella scelta. Ora puoi modificare il modello aprendo e modificando i file estratti. Dopo che hai finito, puoi importare il tuo nuovo modello in KDevelop aprendo il gestore modelli, attivando la scheda appropriata (ne **Modelli di progetto** ne **Modelli di file**) e facendo clic su **Carica modello**. Aprire il file di descrizione del modello, che non è né `.kdevtemplate` né `.desktop`. KDevelop comprimerà i file in un archivio per i modelli e importerà il modello.

> NOTA
> Quando stai copiando un modello esistente, assicurati di rinominarlo prima di importarlo ancora. Altrimenti sovrascriverai il vecchio modello, o finirai con due modelli con lo stesso nome. Per rinominare un modello, rinomina il file di descrizione in qualcosa di unico (ma tieni il suffisso), e cambia la voce `Nome` nel file di descrizione.

Se vuoi scrivere un modello da zero, puoi iniziare con un campione di un modello di una classe C++ da creare un nuovo progetto e selezionare il `Modello di classe C++` per i progetti nella categoria `KDevelop`.

Capitolo 5

Compilare progetti con Makefile personalizzati

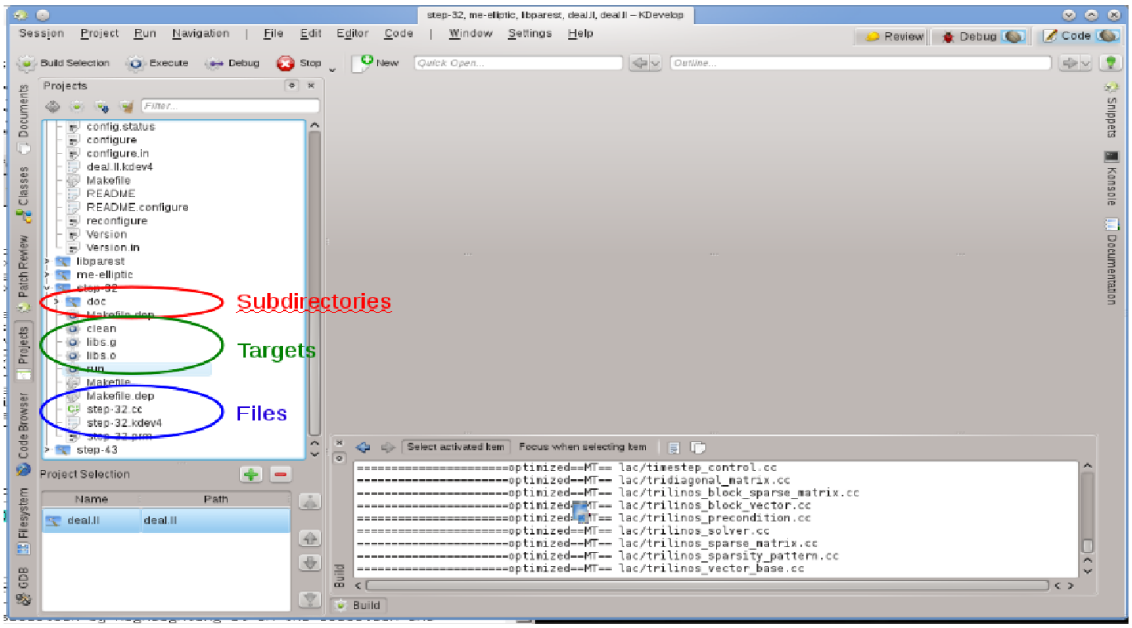
Molti progetti descrivono il modo in cui i file devono essere compilati (e quali file devono essere ricompilati una volta che viene modificato il file sorgente o d'intestazione) usando Makefile che sono interpretati da **make** (vedi, per esempio, GNU make). Per progetti semplici, è spesso molto facile impostare questo file a mano. Progetti più grandi integrano spesso i loro Makefile con **GNU autotools** (autoconf, autoheader, automake). In questa sezione, assumiamo semplicemente di avere un Makefile per il tuo progetto e che tu vuoi istruire KDevelop a interagire con questo.

> NOTA
> KDevelop 4.x non sa molte cose su **GNU autotools** nel periodo in cui questa sezione è stata scritta. Se il tuo progetto li usa, dovrai eseguire a mano su una riga di comando `./configure` o uno qualsiasi degli altri comandi correlati. Se vuoi fare in questo modo usando KDevelop, apri lo strumento **Konsole** (se necessario aggiungilo alla finestra usando il menu **Finestra** → **Aggiungi vista strumento**) che ti fornisce un terminale ed esegui `./configure` dalla linea di comando ottenuta.

Il primo passo consiste nell'istruire KDevelop sugli obiettivi del tuo Makefile. Ci sono due modi di farlo: selezionando obiettivi Makefile individuali, e scegliendo un insieme di obiettivi che potresti compilare di frequente. Per entrambi gli approcci, apri lo strumento **Progetti** facendo clic sul pulsante **Progetti** nella finestra principale di KDevelop (se non hai questo pulsante vedi sopra su come aggiungerlo). La finestra strumento **Progetti** ha due parti: la metà superiore — chiamata **Progetti** — elenca tutto dei tuoi progetti e ti consente di espandere la struttura ad albero sottostante. La metà inferiore — chiamata **Selezione progetto** — elenca un sottoinsieme di quei progetti che saranno compilati se scegli dal menu **Progetto** → **Compila selezione** o premi **F8**; torneremo a questa punto sotto.

5.1 Compilare obiettivi Makefile individuali

Nella parte superiore della vista progetto, espandi il sotto-albero di un progetto, diciamo quello di cui vuoi eseguire un particolare Makefile. Questo ti darà (i) delle cartelle sotto questo progetto, (ii) dei file nella cartella di livello superiore di questo progetto, (iii) dei Makefile che KDevelop può identificare. Queste categorie sono mostrare nella figura a destra. Nota che KDevelop *interpreta* fino ad un certo punto la sintassi di Makefile e perciò ti può offire obiettivi definiti in questo Makefile (anche se questa interpretazione ha i suoi limiti se gli obiettivi sono composti o impliciti).



Per compilare uno dei obiettivi elencati, fai clic con il tasto destro del mouse su di esso e seleziona **Compila**. Per esempio, facendolo con 'make' l'obiettivo semplicemente eseguirà 'make clean'. Puoi vederlo accadere nella sottofinestra che si apre chiamata **Compila**, che mostra il comando e l'output.(Questa finistra corrisponde allo strumento **Compila**, così che tu possa chiudere e riaprire la finestra usando il pulsante dello strumento **Compila**) nella finestra principale. Questo è mostrato in basso a destra nella figura).

5.2 Selezionare una collezione di obiettivi Makefile per compilazioni ripetute

Fare clic destro sul singolo Makefile scelto ogni volta che si desidera compilare qualcosa, è una perdita di tempo. Invece, sarebbe bello avere obiettivi individuali per uno o più progetti della sessione che possiamo compilare più volte senza fare molto lavoro con il mouse. Da ciò viene il concetto di 'Compila selezioni obiettivi': si tratta di una collezione di obiettivi Makefile che sono compilati uno dopo l'altro ogni volta che premi il pulsante **Compila selezione** nell'elenco pulsanti in alto, seleziona la voce del menu **Progetto** → **Compila selezione**, oppure premi il tasto funzione **F8**.

L'elenco degli obiettivi Makefile selezionati è mostrato nella metà inferiore della vista strumento **Progetti**.

Per impostazione predefinita, la selezione contiene tutti i progetti, ma puoi cambiarla. Per esempio, se i tuoi progetti contengono tre progetti (una libreria base L e due applicazioni A e B), ma stai ancora lavorando al progetto A, allora potresti voler rimuovere il progetto B dalla selezione evidenziandolo nella selezione e premendo il pulsante . Inoltre, probabilmente vorrai assicurarti che la libreria L sia compilata prima del progetto A spostando le voci nella selezione in alto o in basso usando i pulsanti a destra dell'elenco. Puoi anche usare un Makefile obiettivo particolare nella selezione facendo clic con il tasto destro su di esso e selezionando **Aggiungi al buildset**, o evidenziandolo e premendo il pulsante  appena sopra agli obiettivi selezionati.

KDevelop ti permette d configurare cosa fare ogni volta che compili la selezione. A tale fine usa la voce del menu **Progetto** → **Apri configurazione**. Lì, puoi selezionare, per esempio, il numero di job che dovrebbe eseguire 'make' — se il tuo computer ha, diciamo, un processore a 8 core,

allora mettere 8 in questo campo sarebbe una scelta utile. In questa finestra di dialogo, l'**obiettivo make predefinito** è un obiettivo Makefile usato per *tutti* gli obiettivi in questa selezione.

5.3 Cosa fare con i messaggi di errore

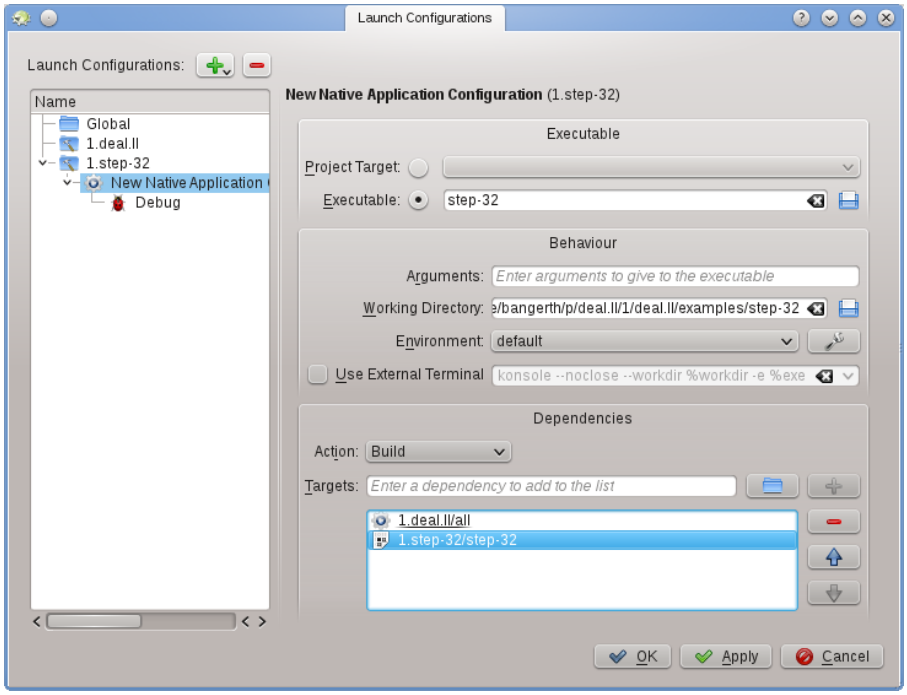
Se il compilatore incontra un messaggio d'errore, fai clic sulla riga con il messaggio di errore e l'editor salterà in quella riga (e se disponibile la colonna) dove è stato segnalato l'errore. A seconda del messaggio di errore, KDevelop potrebbe offrirti diverse possibili azioni per correggere l'errore, per esempio dichiarare una variabile non dichiarata in precedenza se è stato trovato un simbolo sconosciuto.

Capitolo 6

Eseguire programmi in KDevelop

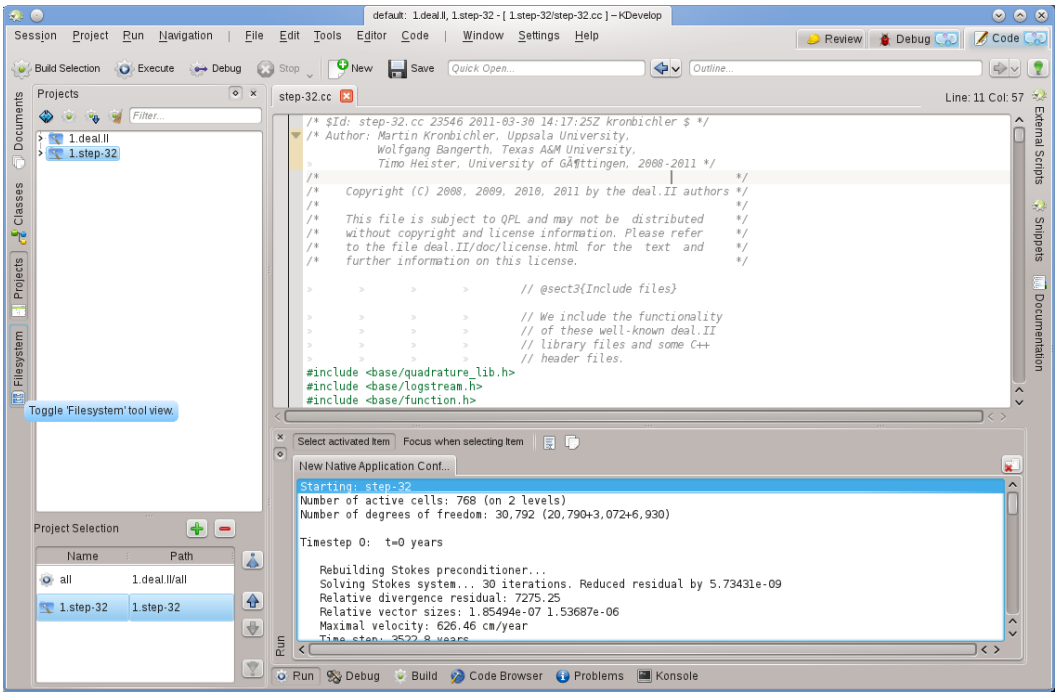
Una volta che hai compilato un programma, lo vorrai eseguire. A tale scopo devi configurare per i tuoi progetti i *Lanci*. Un *Lancio* è formato dal nome di un eseguibile, un insieme di parametri della riga di comando, e un ambiente di esecuzione (ad esempio 'esegui questo programma in una shell', o 'esegui questo programma in un debugger').

6.1 Impostare i lanci in KDevelop



Per impostarli vai alla voce del menu **Esegui** → **Configura i lanci**, e seleziona il progetto a cui vuoi aggiungere un lancio, e fai clic sul pulsante . Poi inserisci il nome dell'eseguibile, e il percorso dove vuoi eseguire il programma. Se l'esecuzione dell'eseguibile dipende dalla compilazione dell'eseguibile e/o da altre librerie, allora potresti volerle aggiungere all'elenco in basso: seleziona **Compila** dal menu a discesa, poi premi il simbolo a destra della casella

di testo e seleziona qualunque obiettivo vuoi compilare. Nell'esempio qui sopra ho selezionato l'obiettivo **tutto** dal progetto `1.deal.II` e `step-32` dal progetto `1.step-32` per assicurarmi che sia la libreria base che il programma siano compilati e aggiornati prima che il programma sia effettivamente eseguito. Già che ci sei, potresti anche configurare il lancio del debug facendo clic sul simbolo **Debug** e aggiungere il nome del debugger; se è il debugger di sistema (ad es. gdb on Linux®), non hai bisogno di fare questo passo.



Ora puoi cercare di eseguire il programma: seleziona dal menu della finestra principale di KDevelop **Esegui** → **Esegui lancio** (o premi **Shift-F9**) e il tuo programma dovrebbe essere eseguito in una sottofinestra separata di KDevelop. La figura qui sopra mostra il risultato: la nuova sottofinestra in basso relativa allo strumento **Esegui** mostra l'output del programma in esecuzione, in questo caso il programma `step-32`.

> NOTA
> Se hai configurato dei lanci multipli, puoi scegliere quale eseguire quando premi **Shift-F9** andando in **Esegui** → **Configurazione di lancio attuale**. Esiste un modo non ovvio per modificare il nome di una configurazione: nella finestra di dialogo che si ottiene quando selezioni **Esegui** → **Configurazione di lancio attuale**, il doppio clic sul nome della configurazione nella vista ad albero a sinistra, ti permetterà di modificare il nome della configurazione.

6.2 Alcune utili scorciatoie da tastiera

Eseguire un programma	
F8	Compila (chiama make)
Shift-F9	Esegui

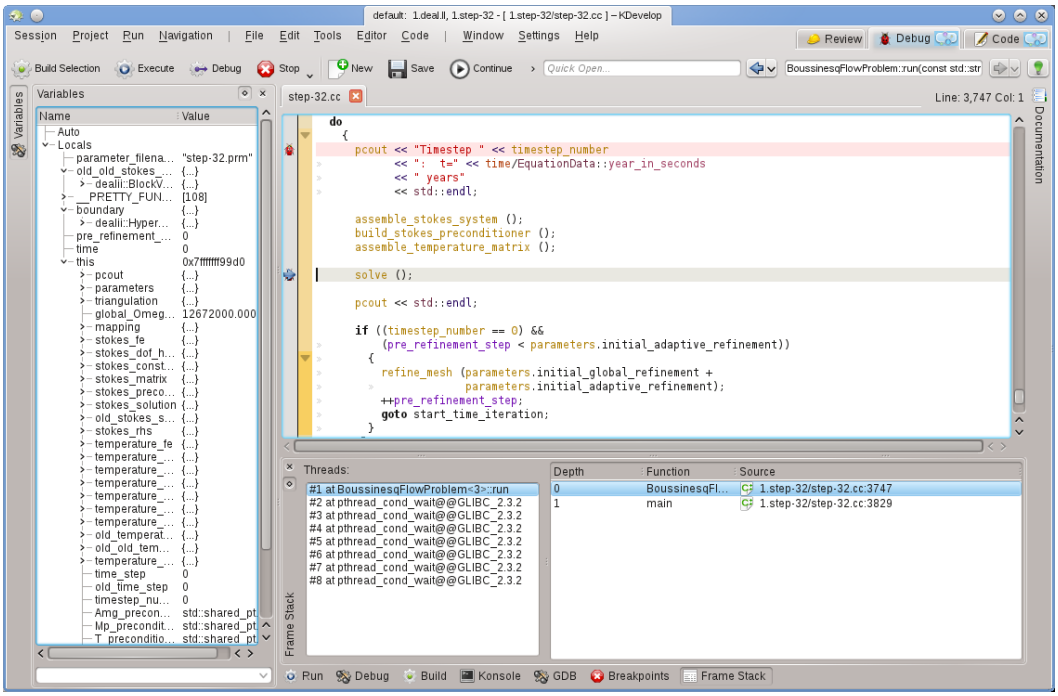
Alt-F9	Esegui il programma nel debugger; potresti voler impostare i breakpoint in anticipo, per esempio facenco clic con il tasto destro del mouse su una riga particolare nel codice sorgente

Capitolo 7

Fare il debug dei programmi in KDevelop

7.1 Eseguire un programma nel debugger

Una volta che hai un lancio configurato (vedi Eseguire programmi), puoi anche eseguirlo in un debugger: seleziona la voce del menu **Esegui** → **Debug del lancio**, o premi **Alt-F9**. Se hai familiarità con gdb, l'effetto è lo stesso di avviare gdb con l'eseguibile specificato nella configurazione di lancio e poi lanciare l'esecuzione con `Esegui`. Questo significa che se il programma chiama da qualche parte la funzione `abort()` (ad es. quando incappi in un'asserzione) o se c'è un segmentation fault, allora il debugger si fermerà. D'altra parte, se il programma viene eseguito fino alla fine (con o senza fare la cosa giusta) allora il debugger non si fermerà da solo prima che il programma sia finito. In quest'ultimo caso, vorrai impostare un breakpoint su tutte quelle righe del tuo codice nelle quali vuoi che il debugger si fermi prima di eseguire il debug del lancio. Questo puoi farlo spostando il cursore del mouse su una riga di quel tipo e selezionando la voce del menu **Esegui** → **Attiva/Disattiva breakpoint**, o facendo clic con il tasto destro su una riga e selezionando dal menu contestuale **Attiva/Disattiva breakpoint**.

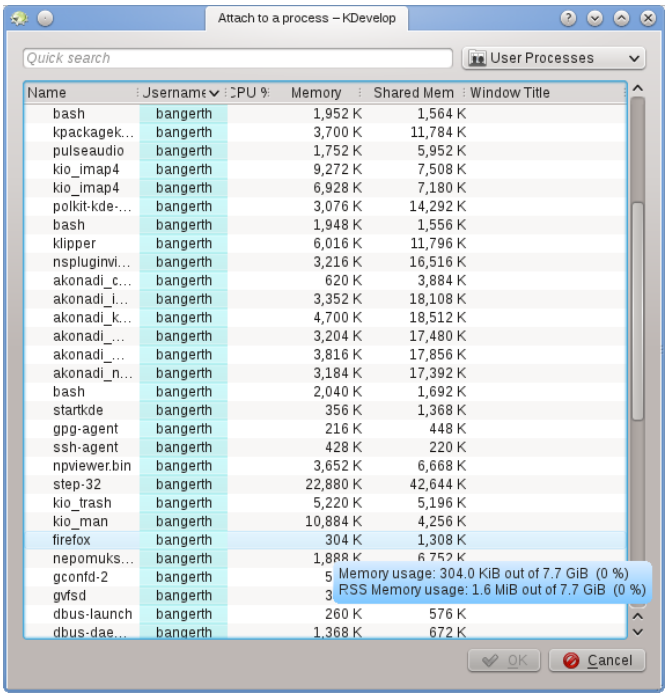


Eseguire un programma nel debugger porrà KDevelop in una diversa modalità: rimpiazzerà tutti i pulsanti 'Strumento' della finestra principale con quelli più appropriati per il debug, piuttosto che per la modifica. Puoi vedere in quale modalità sei guardando in alto a destra nella finestra: ci sono tabelle chiamate **Revisione**, **Debug**, e **Codice**; fare clic su queste ti permette di scegliere una delle tre modalità: ogni modalità ha un proprio insieme di viste strumento, che puoi configurare allo stesso modo in cui hai configurato lo strumento **Codice** nella sezione Strumenti e viste.

Una volta che il debugger si ferma (ad un breakpoint, o in un punto dove viene chiamata la funzione `abort()`) puoi controllare una serie di informazioni riguardo il tuo programma. Per esempio, nella figura qui sopra, abbiamo selezionato lo strumento in basso **Frame Stack** (più o meno equivalente ai comandi 'backtrace' e 'info threads' di gdb) che mostra, sulla sinistra, i vari thread che sono attualmente in esecuzione nel tuo programma (qui in totale 8) e come, sulla destra, l'esecuzione ha raggiunto il punto di arresto (qui: `main()` called `run()`; l'elenco sarebbe più lungo se ci fossimo fermati su una funzione chiamata dalla stessa funzione `run()`). Sulla sinistra, possiamo controllare le variabili locali e l'oggetto corrente (l'oggetto puntato dalla variabile `this`).

Da qui ci sono diverse cose che puoi fare: puoi eseguire la riga corrente (**F10**, il comando 'successivo' di gdb), passare dentro le funzioni (**F11**, il comando 'passo' di gdb), o eseguire la funzione fino alla fine (**F12**, il comando 'ferma' di gdb). In ogni fase, KDevelop aggiorna le variabili mostrate sulla sinistra con i valori attuali. Puoi anche passare il mouse sopra un simbolo del tuo codice, ad es. una variabile; KDevelop mostrerà quindi il valore attuale di quel simbolo e chiederà di fermare il programma durante l'esecuzione la prossima volta che il valore di questa variabile cambierà. Se conosci gdb, puoi anche fare clic sul pulsante strumento **GDB** in basso e avere la possibilità di inserire comandi gdb, per esempio al fine di cambiare il valore di una variabile (per la quale attualmente non sembra esserci altro modo).

7.2 Collegare il debugger ad un processo in esecuzione



A volte si vuole fare il debug di un programma che è già in esecuzione. Uno scenario di questo è il debug di programmi paralleli usando MPI, o per fare il debug di un processo in esecuzione da molto tempo in background. Per farlo, andare alla voce del menu **Esegui** → **Collega al debugger**, questo aprirà una finestra come quella qui sopra. Vorrai selezionare il programma che corrisponde al progetto che hai aperto in questo momento in KDevelop - nel mio caso questo sarebbe il programma step-32.

L'elenco dei programmi può confondere perché è spesso lungo come nel caso mostrato qui. Puoi rendere la tua vita un po' più semplice andando nella casella a tendina in alto a destra della finestra. Il valore predefinito è **Processi utente**, cioè tutti i programmi in esecuzione degli utenti connessi al momento al computer (se questo è il tuo desktop o laptop, probabilmente sei l'unico utente, oltre gli account root e degli altri servizi); l'elenco, comunque, non include i processi eseguiti dall'utente root. Puoi limitare l'elenco per entrambi scegliendo **Processi propri**, rimuovendo tutti i programmi eseguiti dagli altri utenti. O meglio ancora: seleziona **Solo programmi**, questo rimuove molti processi che sono formalmente in esecuzione sotto il tuo nome ma con i quali non interagisci di solito, come il gestore delle finestre, le attività in background e così via, che sono candidati improbabili per il debug.

Una volta che hai selezionato un processo, collegandolo ti porterà nella modalità di debug di KDevelop, aprendo tutte le solite viste strumento del debugger e fermando il programma nella posizione in cui sembrava essere quando lo hai collegato. Poi potresti voler impostare i breakpoint, i punti di vista, o qualsiasi altra cosa è necessaria e continuare l'esecuzione del programma andando alla voce del menu **Esegui** → **Continua**.

7.3 Alcune utili scorciatoie da tastiera

Debugging	
F10	Passa oltre ('successivo' di gdb)

F11	Passa dentro (‘passo’ di gdb)
F12	Esci da (‘finisci’ di gdb)

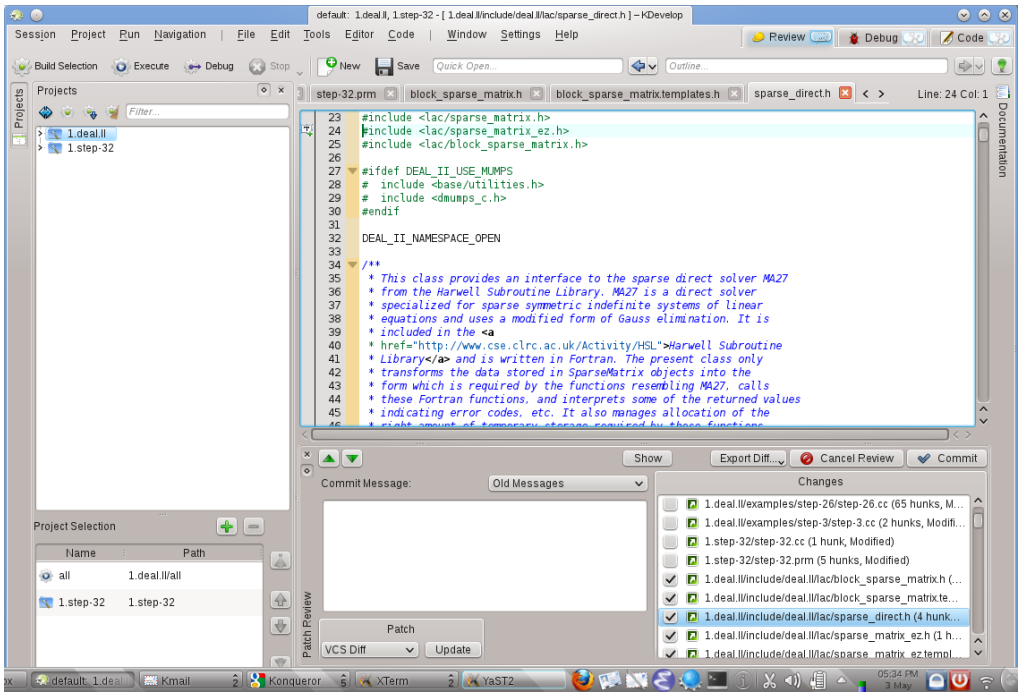
Capitolo 8

Lavorare con i sistemi di controllo versione

Se stai lavorando su progetti di grandi dimensioni, è probabile che il codice sorgente sia gestito da un sistema di controllo versione come subversion o git. La descrizione seguente è scritta prendendo in considerazione **subversion** ma sarà altrettando vera se si usa **git** o qualsiasi altro sistema di controllo versione.

Per prima cosa, non è che se la cartella in cui si trova un progetto è sotto controllo versione, automaticamente KDevelop lo rileva. In altre parole: non è necessario dire a KDevelop di andare a prelevare un copia durante l'impostazione del progetto; ma è una buona scelta far puntare KDevelop a una cartella nella quale si è in precedenza scaricato un progetto dopo averlo prelevato dal deposito. Se si ha sotto controllo versione una cartella di questo tipo, aprire la vista strumento **Progetti**. Ci sono poi una serie di cose che si possono fare:

- Se la cartella è diventata obsoleta, la puoi aggiornare dal deposito: fai clic col tasto destro del mouse sul nome del progetto, vai al menu **Subversion** e seleziona **Aggiorna**. Questo aggiornerà tutti i file che appartengono a questo progetto a quelli presenti nel deposito.
- Se vuoi limitare questa azione alle singole sottocartelle o file, allora espandi la vista ad albero del progetto al livello che vuoi e fai clic destro sul nome della sotto cartella o file, quindi fai lo stesso di sopra.



- Se hai modificato uno o più file, espandi la vista di questo progetto alla cartella in cui questi file sono presenti e fai clic destro sulla cartella. Questo ti darà una voce del menu **Subversion** che ti proporrà diverse scelte. Scegli **Mostra differenze** per vedere le differenze tra la versione che hai modificato e la versione nel deposito che hai prelevato in precedenza (la revisione 'base'). La vista risultante mostrerà le 'differenze' per tutti i file della cartella.
- Se hai modificato solo un file, puoi anche ottenere il menu **Subversion** per questo file facendo semplicemente clic sul nome corrispondente nella vista progetto. In modo più semplice, basta fare clic destro nella vista **Editor** nella quale hai aperto questo file ottenendo anche questa opzione del menu.
- Se vuoi inviare uno o più file modificati, fai clic destro o su un singolo file, sottocartella o su tutto il progetto e seleziona **Subversion** → **Deposita**. Questo ti porterà nella modalità **Revisione**, la terza modalità oltre a **Codice** e **Debug** nell'angolo in alto a destra della finestra principale di KDevelop. La figura sulla destra mostra com'è. Nella modalità **Revisione**, la parte superiore mostra le differenze per l'intera/o sottocartella/progetto e ciascuno dei singoli file modificati con le modifiche evidenziate (vedi le varie tabelle di questa parte della finestra). Per impostazione predefinita, tutti i file modificati sono nel changeset che stai per depositare, ma puoi deselezionarne alcuni se le loro modifiche non sono legate a cosa vuoi depositare. Per esempio, nell'esempio sulla destra ho deselezionato `step-32.cc` e `step-32.prm` perché le modifiche in questi file non hanno niente a che fare con le altre che ho fatto nel progetto e che non voglio ancora depositare (potrei volerlo fare in un secondo momento). Dopo aver revisionato le modifiche puoi inserire un messaggio di deposito nella casella di testo e premere **Deposita** sulla destra per inviare quello che vuoi.
- Proprio come nella visualizzazione delle differenze, se vuoi inviare un unico file è sufficiente fare clic destro nella finestra dell'editor per avere la voce del menu **Subversion** → **Deposita**.

Capitolo 9

Personalizzare KDevelop

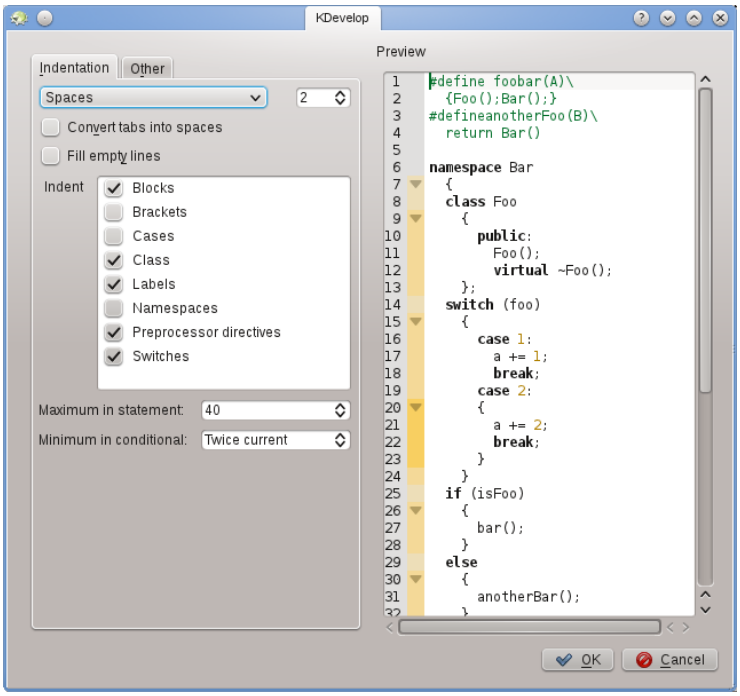
Quando vuoi cambiare l'aspetto o il comportamento predefinito di KDevelop, per esempio perché sei abituato a differenti scorciatoie da tastiera o perché il tuo progetto richiede uno stile di identazione del codice sorgente diverso. Nelle sezioni seguenti, discuteremo brevemente i diversi modi in cui è possibile personalizzare KDevelop per questi scopi.

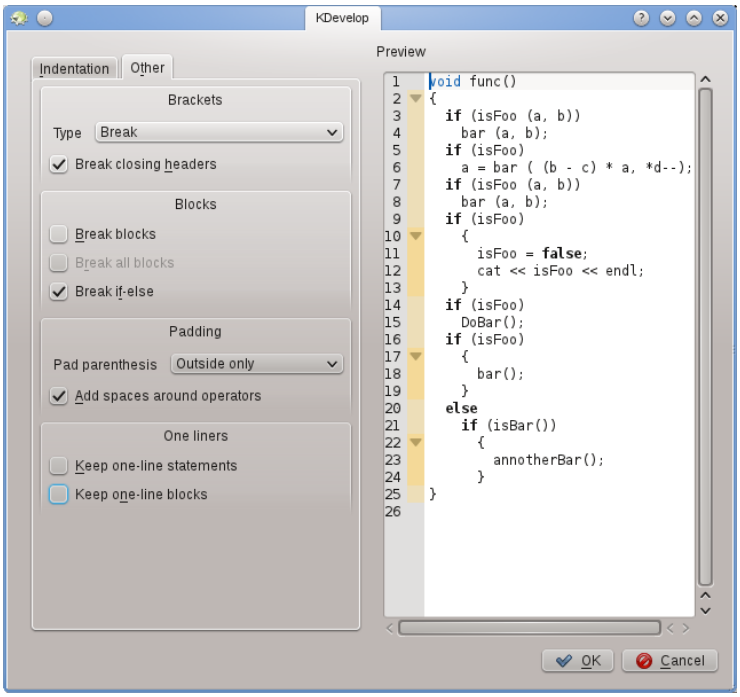
9.1 Personalizzare l'editor

Ci sono molte cose utili che si possono configurare nell'editor di KDevelop. L'uso più comune consiste nell'abilitare la numerazione delle righe usando la voce del menu **Editor** → **Vista** → **Mostra i numeri di riga**, il che rende più facile abbinare i messaggi di errore del compilatore o i messaggi di debug con le posizioni nel codice. Nello stesso sottomenu potresti voler anche abilitare il *bordo delle icone* - una colonna alla sinistra del codice nella quale KDevelop mostrerà le icone come ad esempio quella di un breakpoint su una riga.

9.2 Personalizzare l'identazione del codice

A molti di noi piace il testo formattato in un modo particolare. Molti progetti obbligano ad usare un particolare stile di identazione. Alcuni di questi potrebbero non corrispondere agli stili predefiniti di KDevelop. Tuttavia, questo può essere personalizzato: andare alla voce del menu **Impostazioni** → **Configurare KDevelop**, poi fare clic su **Formattatore sorgenti** sulla sinistra. È possibile scegliere uno degli stili predefiniti di identazione, che sono quelli tra i più usati, o definirne uno proprio aggiungendo un nuovo stile e poi modificandolo. Potrebbe non esserci un modo di ricreare esattamente lo stile di identazione usato nei sorgenti di un progetto passato, ma ti ci puoi avvicinare usando le impostazioni di un nuovo stile; un esempio è mostrato nelle due figure qui sotto.


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

NOTA
Con **KDevelop 4.2.2**, puoi creare un nuovo stile per un particolare tipo MIME (ad es. per i file di intestazione C++) ma questo stile potrebbe non apparire nella lista dei possibili stili per altri tipi MIME (ad es. per i file sorgente C++) anche se potrebbe essere utile usare lo stesso stile per entrambi i tipi di file. Perciò dovrai definire lo stile due volte, una volta per il file di intestazione e l'altra per i file sorgente. Questo comportamento è stato riportato come bug 272335 di KDevelop.

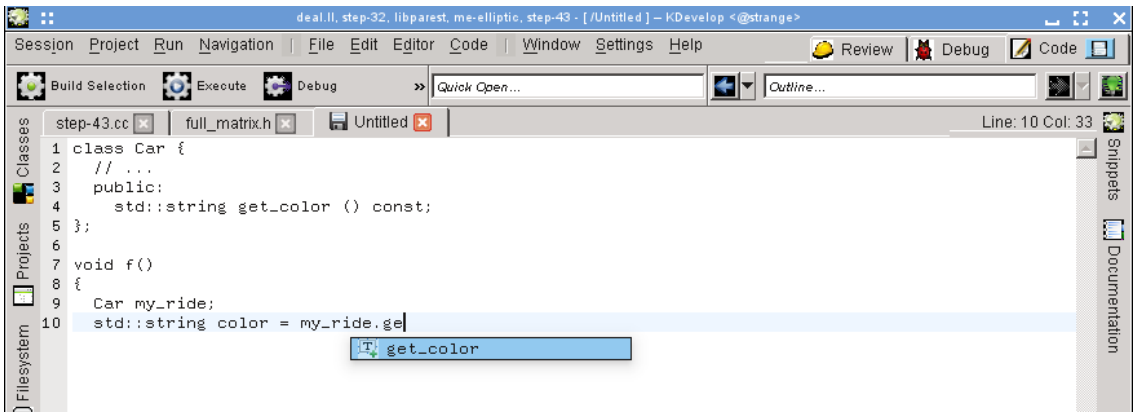
9.3 Personalizzare le scorciatoie da tastiera

KDevelop ha una lista quasi infinita di scorciatoie da tastiera (alcune di esse sono elencate nelle 'Sezioni scorciatoie da tastiera utili' di diversi capitoli di questo manuale) che possono essere modificate a piacere attraverso il menu **Impostazioni** → **Configura le scorciatoie**. Nella parte superiore della finestra di dialogo è possibile inserire una parola, una volta inserita saranno mostrate solo le scorciatoie corrispondenti alla parola, quindi è possibile modificare la combinazione dei tasti da associare alla scorciatoia.

Due che sono state considerate utili da cambiare riguardano l'impostazione di **Align** per il **Tab** (molte persone non inseriscono i tab a mano e preferiscono piuttosto che l'editor scelga il layout del codice; con la scorciatoia modificata, la pressione di **Tab** fa sì che KDevelop identi/deidenti/allinei il codice). La seconda è mettere **Attiva/disattiva breakpoint** su **Ctrl-B** perché è una operazione abbastanza comune.

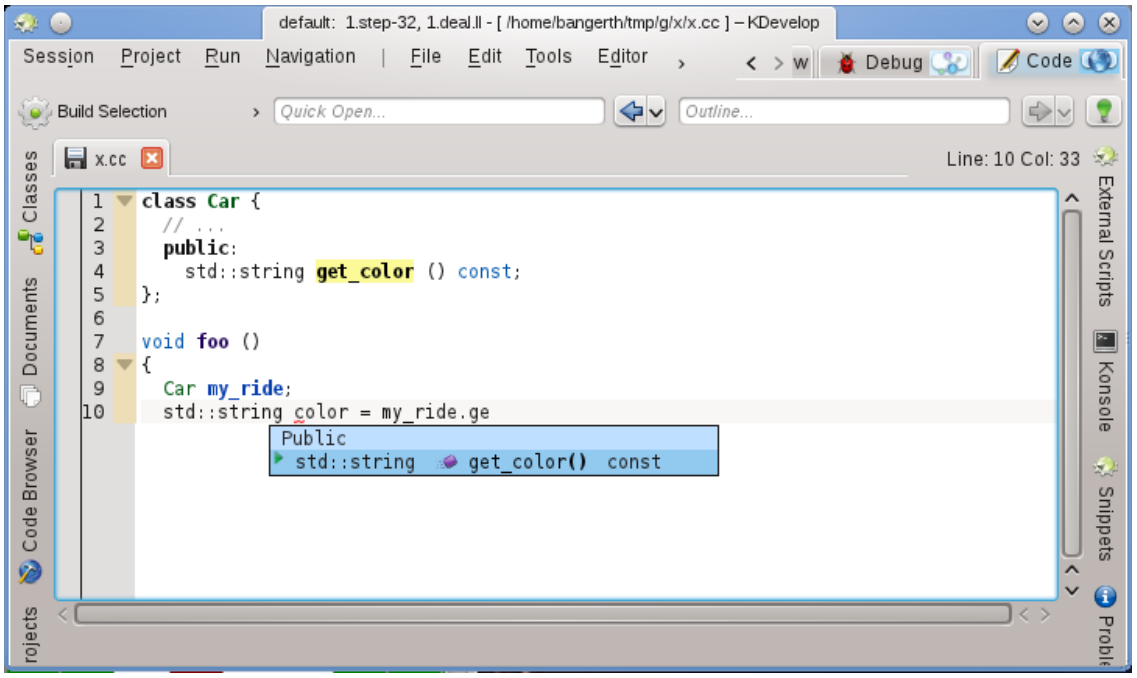
9.4 Personalizzare l'auto-completamento del codice

Il completamento del codice è discusso nella sezione che riguarda la scrittura del codice sorgente di questo manuale. In KDevelop, viene da due fonti: l'editor, e il motore di analisi. L'editor (Kate) è una componente dell'ambiente KDE e offre l'auto-completamento basato sulle parole già viste in altre parti dello stesso documento. L'auto-completamento può essere identificato nel suggerimento dall'icona che lo precede:



Il completamento del codice dell'editor può essere personalizzato con **Impostazioni** → **Configura editor** → **Modifica** → **Completamento delle parole**. In particolare, puoi selezionare quanti caratteri di una parola è necessario scrivere prima che questa venga completata.

Però l'auto-completamento di KDevelop è molto più potente in quanto tiene conto delle informazioni semantiche del contesto. Per esempio, sa quali funzioni membro proporre quando digiti `object.`, ecc., come mostrato qui:



Queste informazioni sul contesto provengono dalle varie estensioni di supporto linguaggio, che possono essere usate dopo che è stato salvato un dato file (in modo da poter poi controllare il tipo di file e utilizzare il supporto linguaggio corretto)

Il completamento di KDevelop è impostato per comparire durante la digitazione, subito, praticamente ovunque ci sia bisogno di completare qualcosa. Questo è configurabile in **Impostazioni** → **Configura KDevelop** → **Supporto linguaggio**. Se non è già impostato (come dovrebbe, per impostazione predefinita), assicurarsi che sia impostato **Abilita l'invocazione automatica**.

KDevelop ha due modi per visualizzare un completamento: il **Completamento automatico minimo** che mostra solo le informazioni base in suggerimenti di completamento (cioè i namespace, le classi, le funzioni, o nomi delle variabili). Questo sarà simile al completamento di Kate (eccetto le icone).

Però, il **Completamento totale** in aggiunta mostrerà il tipo di ogni voce, e in caso di funzioni, anche gli argomenti che prende. Inoltre se stai attualmente inserendo gli argomenti di una funzione, il completamento totale avrà un riquadro di informazioni aggiuntivo sopra il cursore che ti mostrerà l'argomento su cui stai lavorando adesso.

Il completamento del codice di KDevelop dovrebbe evidenziare in verde tutti gli elementi di completamento che corrispondono al tipo attualmente previsto sia nel completamento minimo che totale, noto come 'best-matches'.

Le tre possibili scelte per il livello di completamento nella finestra di dialogo della configurazione sono:

- **Sempre completamento minimo**: non mostra mai il 'Completamento totale'
- **Completamento automatico minimo**: mostra solo il 'Completamento totale' quando l'autocompletamento è stato avviato automaticamente (cioè, ogni volta che premi **Ctrl-Spazio**)
- **Sempre completamento totale**: mostra sempre il 'Completamento totale'

Capitolo 10

Riconoscimenti e licenza

Simone Solinas ksolsim@gmail.com