

# Manual de KDevelop

Esta documentación ha sido convertida a partir de la página  
KDevelop4/Manual de KDE UserBase.  
Traductor: Eloy Cuadra



# Manual de KDevelop

# Índice general

<b>1. ¿Qué es KDevelop?</b>	<b>6</b>
<b>2. Sesiones y proyectos: lo básico de KDevelop</b>	<b>8</b>
2.1. Terminología . . . . .	8
2.2. Configuración de una sesión e importación de un proyecto existente . . . . .	9
2.2.1. Opción 1: importar un proyecto desde un servidor de un sistema de control de versiones . . . . .	9
2.2.2. Opción 2: importar un proyecto que ya esté en su disco duro . . . . .	10
2.3. Configuración de una aplicación como segundo proyecto . . . . .	10
2.4. Creación de proyectos desde cero . . . . .	10
<b>3. Trabajar con el código fuente</b>	<b>12</b>
3.1. Herramientas y vistas . . . . .	12
3.2. Exploración del código fuente . . . . .	14
3.2.1. Información local . . . . .	14
3.2.2. Información del ámbito de los archivos . . . . .	16
3.2.3. Información del ámbito del proyecto y de la sesión . . . . .	17
3.2.4. El resaltado con los colores del arcoíris explicado . . . . .	19
3.3. Navegar por el código fuente . . . . .	19
3.3.1. Navegación local . . . . .	19
3.3.2. Navegación en el ámbito de archivos y modo de esquema . . . . .	20
3.3.3. Navegación en el ámbito del proyecto y de la sesión: navegación semántica	21
3.4. Escritura de código fuente . . . . .	25
3.4.1. Terminación automática . . . . .	25
3.4.2. Añadir nuevas clases e implementar funciones miembro . . . . .	27
3.4.3. Documentar declaraciones . . . . .	31
3.4.4. Cambiar el nombre de variables, funciones y clases . . . . .	34
3.4.5. Fragmentos de código . . . . .	35
3.5. Modos y conjuntos de trabajo . . . . .	37
3.6. Algunos atajos de teclado útiles . . . . .	39

<b>4. Generación de código con plantillas</b>	<b>41</b>
4.1. Creación de una nueva clase . . . . .	41
4.2. Creación de una nueva prueba unitaria . . . . .	43
4.3. Otros archivos . . . . .	43
4.4. Gestión de plantillas . . . . .	44
<b>5. Construir (compilar) proyectos con Makefiles personalizados</b>	<b>46</b>
5.1. Construcción de objetivos Makefile individuales . . . . .	46
5.2. Selección de una colección de Makefiles de destino para compilación repetitiva . .	47
5.3. Qué hacer con los mensajes de error . . . . .	48
<b>6. Ejecutar programas en KDevelop</b>	<b>49</b>
6.1. Configuración de lanzadores en KDevelop . . . . .	49
6.2. Algunos atajos de teclado útiles . . . . .	50
<b>7. Depuración de programas en KDevelop</b>	<b>52</b>
7.1. Ejecutar un programa en el depurador . . . . .	52
7.2. Adjuntar el depurador a un proceso en ejecución . . . . .	54
7.3. Algunos atajos de teclado útiles . . . . .	54
<b>8. Trabajar con sistemas de control de versiones</b>	<b>56</b>
<b>9. Personalización de KDevelop</b>	<b>58</b>
9.1. Personalización del editor . . . . .	58
9.2. Personalización de la sangría del código . . . . .	58
9.3. Personalización de los atajos de teclado . . . . .	60
9.4. Personalización de la terminación automática de código . . . . .	60
<b>10. Créditos y licencia</b>	<b>62</b>

## **Resumen**

KDevelop es un entorno de desarrollo integrado que se puede usar para una amplia variedad de tareas de programación.

## Capítulo 1

# ¿Qué es KDevelop?

**KDevelop** es un moderno entorno de desarrollo integrado (IDE) para C++ (y otros lenguajes) que es una de las muchas [aplicaciones de KDE](#). Como tal, funciona sobre Linux® (incluso si está utilizando cualquier otro escritorio, como GNOME), aunque también está disponible para otras variantes de UNIX®, e incluso para Windows.

KDevelop ofrece todos los servicios de un IDE moderno. Para grandes proyectos y aplicaciones, la característica más importante es que KDevelop *entiende* C++: analiza todo el código fuente y recuerda las funciones miembro que posee cada clase, dónde se han definido las variables, cuáles son sus tipos y muchas otras cosas de su código fuente. Por ejemplo, si uno de los archivos de cabecera de su proyecto declara una clase

```
class Car {
    // &#8230;
    public:
        std::string get_color () const;
};
```

y posteriormente escribe en su programa

```
Car my_ride;
// &#8230;hacer algo con esta variable&#8230;
std::string color = my_ride.ge
```

recordará que el `my_ride` de la última línea es una variable de tipo `Car` y le ofrecerá la posibilidad de completar `ge` como `get_color()`, ya que esta es la única función miembro de la clase `Car` que empieza de ese modo. En lugar de continuar escribiendo, solo tiene que pulsar **Intro** para obtener la palabra completa; esto le ahorra pulsaciones de teclado, le evita errores al teclear y no le obliga a recordar los nombres exactos de los cientos o miles de funciones y clases que componen los grandes proyectos.

Como segundo ejemplo, supondremos que tiene un código como este:

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

## Manual de KDevelop

Si sitúa el cursor del ratón sobre el símbolo `var` de la función `bar` se le ofrecerá la posibilidad de ver todos los usos de este símbolo. Si pulsa sobre esta opción le mostrará únicamente los usos de esta variable dentro de la función `bar`, ya que KDevelop entiende que la variable `var` de la función `foo` no tiene nada que ver con ella. De igual modo, si pulsa con el botón derecho del ratón sobre el nombre de la variable podrá cambiar el nombre de la misma; si lo hace, solo se modificará la variable de la función `bar`, pero no la que tiene el mismo nombre dentro de la función `foo`.

Pero KDevelop no es solamente un editor de código inteligente; hay otras cosas que KDevelop hace bien. Por supuesto, puede resaltar el código fuente con diferentes colores, posee sangrado personalizable, dispone de una interfaz integrada con el depurador `gdb` de GNU, puede mostrarle información sobre una función al situar el cursor del ratón sobre ella, puede lidiar con diferentes clases de entornos de construcción y compiladores (por ejemplo con proyectos basados en `make` y `cmake`), entre otras muchas cosas interesantes que se describen en este manual.

## Capítulo 2

# Sesiones y proyectos: lo básico de KDevelop

En esta sección trataremos sobre la terminología de cómo KDevelop ve el mundo y cómo estructura el trabajo. En particular, introduciremos los conceptos de *sesiones* y *proyectos*, y explicaremos cómo puede configurar los proyectos sobre los que vaya a trabajar usando KDevelop.

### 2.1. Terminología

KDevelop usa los conceptos de *sesiones* y *proyectos*. Una sesión contiene todos los proyectos que están relacionados entre sí. En los ejemplos que siguen supondremos que usted es el desarrollador de una biblioteca y de una aplicación que la usa. Puede pensar que la primera equivale a las bibliotecas centrales de KDE y que la segunda es KDevelop. Otro ejemplo similar sería el de un programador del kernel de Linux<sup>®</sup> que también está trabajando en un controlador de dispositivo para Linux<sup>®</sup> que aún no ha sido integrado en el árbol de código fuente del kernel.

Si tomamos este último caso como ejemplo, tendríamos una sesión en KDevelop que contendría dos proyectos: el del kernel de Linux<sup>®</sup> y el del controlador de dispositivo. Seguramente desearía agrupar ambos proyectos en una única sesión (en lugar de tener dos sesiones con un único proyecto cada una), ya que le resultaría útil poder ver las funciones y las estructuras de datos del kernel en KDevelop mientras escribe el código fuente para el controlador (de este modo, podrá disponer de expansión automática de nombre de funciones y de variables del kernel, o consultar la documentación de cualquier función del kernel mientras está programando el controlador de dispositivo).

Ahora imagine que también es un desarrollador de KDE. Ahora también tendría una segunda sesión que contuviera KDE como proyecto. En principio, sería posible tener una única sesión para todo esto, pero no existe un motivo razonable para ello: cuando trabaja en KDE no necesita acceder a funciones del kernel ni del controlador de dispositivo, y tampoco querrá que se expandan automáticamente los nombres de las clases de KDE cuando trabaje con el kernel de Linux<sup>®</sup>. Del mismo modo, la construcción de algunas de las bibliotecas de KDE es independiente de la recompilación del kernel de Linux<sup>®</sup> (independientemente de que cuando compile el controlador de dispositivo sea una buena idea volver a compilar el kernel de Linux<sup>®</sup> si ha modificado alguno de los archivos de cabecera del kernel).

Para finalizar, otro uso de las sesiones sería cuando está trabajando tanto en la versión de desarrollo actual de un proyecto como en una rama del mismo. En este caso no sería deseable que KDevelop confundiera las clases que pertenecen a la versión principal y a la rama, por lo que tendría dos sesiones con el mismo conjunto de proyectos, pero usando diferentes directorios (que se corresponderían con las diferentes ramas de desarrollo).

## 2.2. Configuración de una sesión e importación de un proyecto existente

Quedémonos con el ejemplo del kernel de Linux<sup>®</sup> y del controlador de dispositivo (tal vez quiera sustituir su propio conjunto de bibliotecas o de proyectos para estos dos ejemplos). Para crear una nueva sesión que contenga estos dos proyectos, vaya a la opción del menú **Sesión** → **Iniciar nueva sesión** (o, si esta es la primera vez que está usando KDevelop, utilice simplemente la sesión predeterminada que se obtiene al iniciar el programa, que estará vacía).

A continuación vamos a añadir proyectos a esta sesión. Por ahora, supondremos que ya existen en algún lugar (el caso de iniciar proyectos desde cero se trata en otra parte de este manual). Para ello, disponemos esencialmente de dos métodos dependiendo de si el proyecto está ya en algún lugar de su disco duro o si es necesario descargarlo de un servidor.

### 2.2.1. Opción 1: importar un proyecto desde un servidor de un sistema de control de versiones

En primer lugar, supongamos que el proyecto que queremos configurar (el kernel de Linux<sup>®</sup>) reside en un sistema de control de versiones de un servidor, del que aún no lo ha descargado a su disco duro. En este caso, vaya al menú **Proyecto** para crear un proyecto para el kernel de Linux<sup>®</sup> dentro de la sesión actual y siga luego estos pasos:

- Vaya a **Proyecto** → **Obtener proyecto** para importar un proyecto.
- Ahora tendrá diversas opciones para iniciar un nuevo proyecto en la sesión actual dependiendo de dónde procedan los archivos de código fuente: puede hacer que KDevelop apunte a un directorio existente (vea la opción 2 más adelante) o puede pedirle a KDevelop que obtenga el código fuente de un repositorio.
- Suponiendo que aún no ha descargado una versión:
  - En el diálogo, debajo de **Seleccione el origen**, elija el uso de **Del sistema de archivos**, **Subversion**, **Git**, **GitHub** o **KDE**.
  - Elija un directorio de trabajo como destino en el que descargar el código fuente
  - Escoja una URL para la ubicación del repositorio desde donde se obtendrán los archivos de código fuente
  - Pulse **Obtener**. Esta acción puede necesitar bastante tiempo dependiendo de la velocidad de su conexión y del tamaño del proyecto. Desafortunadamente, la barra de progreso de KDevelop 4.2.x no muestra ninguna información, pero puede seguir el progreso mirando de vez en cuando la salida de la orden de consola

```
du -sk /ruta/al/proyecto/de/KDevelop
```

para ver la cantidad de datos que ya han sido descargados.

#### NOTA

El problema de la barra de progreso ha sido notificado como [error 256832 de KDevelop](#).

#### NOTA

Durante este proceso también obtenemos el mensaje de error *Necesita especificar una ubicación válida para el proyecto*, que puede ignorar sin mayor problema.

- Le pide que seleccione un archivo de proyecto de KDevelop en este directorio. Como es probable que todavía no tenga uno, pulse **Siguiente**
- Pulse **Siguiente** otra vez
- KDevelop le solicitará entonces que escoja un gestor de proyecto. Si este proyecto usa archivos «make» estándares de UNIX®, escoja el gestor de proyectos para «makefiles» personalizados
- KDevelop comenzará entonces a analizar todo el proyecto. De nuevo, tardará un tiempo para recorrer todos los archivos e indexar sus clases, etc. En la parte inferior izquierda de la ventana principal existe una barra de progreso que muestra cuánto está tardando este proceso (si dispone de varios procesadores podrá acelerar este proceso yendo a la opción del menú **Preferencias** → **Configurar KDevelop**, seleccionando después la página **Analizador en segundo plano** en la parte izquierda e incrementando el número de hilos para en análisis en segundo plano en la parte de la derecha del diálogo).

### 2.2.2. Opción 2: importar un proyecto que ya esté en su disco duro

De forma alternativa, si el proyecto en el que quiere trabajar ya está en su disco duro (por ejemplo, bien porque lo había descargado como un archivo «tar» desde un servidor FTP, bien porque ya había descargado una versión del proyecto desde un sistema de control de versiones, o bien porque se trata de su propio proyecto que existe *solo* en su disco duro), use la opción del menú **Proyecto** → **Abrir/importar proyecto** y, en el diálogo que se le mostrará, escoja el directorio en el que reside su proyecto.

## 2.3. Configuración de una aplicación como segundo proyecto

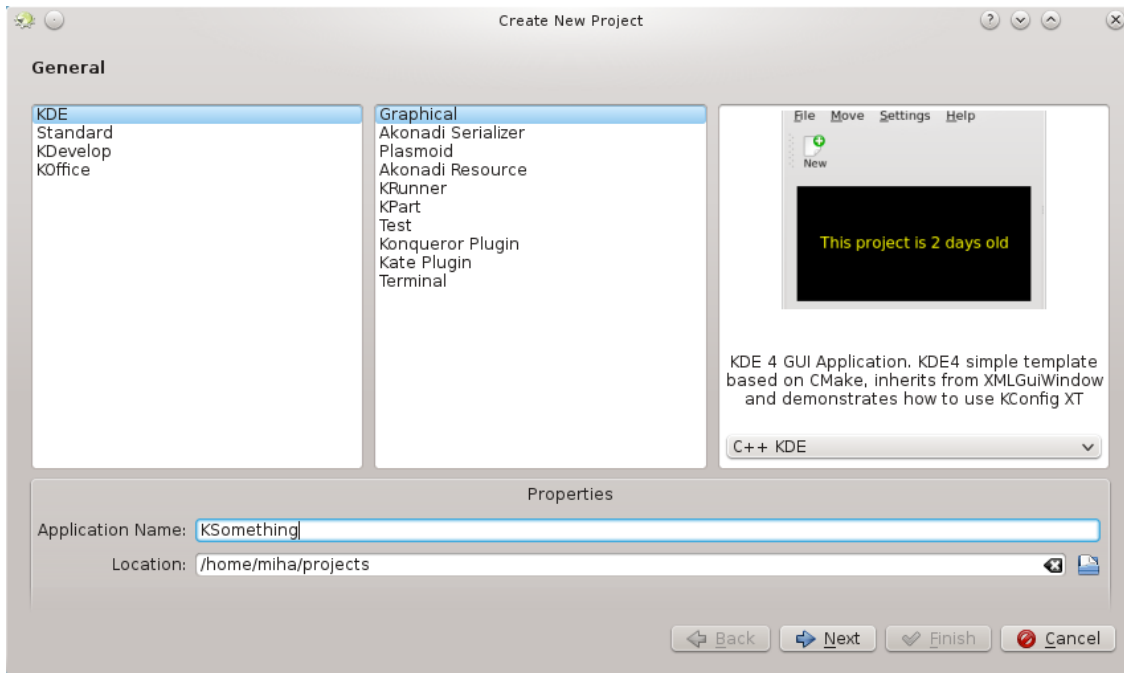
Lo siguiente que querrá hacer será configurar otros proyectos en la misma sesión. En el ejemplo anterior, querría añadir el controlador de dispositivo como segundo proyecto, lo que podrá hacer siguiendo exactamente los mismos pasos.

Si tiene diversas aplicaciones o bibliotecas, solo tiene que repetir los mismos pasos para añadir más proyectos a su sesión.

## 2.4. Creación de proyectos desde cero

Por supuesto, también existe la posibilidad de que quiera comenzar un nuevo proyecto desde cero. Para ello, puede usar la opción del menú **Proyecto** → **Nuevo desde plantilla...**, que le mostrará un diálogo para seleccionar una plantilla. Algunas de estas plantillas de proyecto se proporcionan con KDevelop, pero dispondrá de más si instala la aplicación KAppTemplate. Elija el tipo de proyecto y el lenguaje de programación en el diálogo, introduzca un nombre y la ubicación del proyecto, y luego pulse **Siguiente**.

## Manual de KDevelop



La segunda página del diálogo le permitirá decidir por un sistema de control de versiones. escoja el sistema que desee usar y rellene la configuración específica del sistema, si es necesario. Si no quiere usar ningún sistema de control de versiones, o si pretende hacerlo más adelante de forma manual, elija **Ninguno**. Cuando haya terminado, pulse el botón **Finalizar**.

Su proyecto se creará, de modo que podrá probar a compilarlo o a instalarlo. Algunas plantillas incluirán comentarios dentro del código, e incluso un archivo README independiente, por lo que se le recomienda que lea dicha información en primer lugar. A continuación podrá comenzar a trabajar en su proyecto añadiéndole las funcionalidades que desee.



sobre ellos, se expanden en una ventana secundaria (una *vista*) dentro de la ventana principal. Si vuelve a pulsar sobre el botón, la ventana secundaria desaparecerá.

También puede pulsar la x de la esquina superior derecha de una ventana secundaria para hacer que desaparezca.

La imagen inferior muestra una determinada selección de herramientas, alineadas sobre los márgenes izquierdo y derecho. En la imagen, la herramienta **Clases** está abierta a la izquierda, y la herramienta **Fragmentos** a la derecha, junto con un editor que contiene un archivo de código fuente en la parte central. En la práctica, la mayor parte del tiempo solo tendrá abierto el editor y tal vez una de las herramientas **Clases** o **Navegador de código** a la izquierda. Es muy probable que el resto de vistas de herramientas estén abiertas de forma temporal cuando las esté usando, dejando más espacio para el editor la mayor parte del tiempo.

Cuando ejecute KDevelop por primera vez, debería tener el botón de la herramienta **Proyectos**. Pulse sobre él para abrir una ventana secundaria que muestra los proyectos que ha añadido a la sesión en la parte inferior, junto a una vista del sistema de archivos de los directorios de sus proyectos en la parte superior.

Existen más herramientas que puede usar con KDevelop, aunque no todas ellas están inicialmente presentes como botones en los laterales. Para añadir alguna de ellas, vaya a la entrada del menú **Ventana** → **Añadir vista de herramientas**. Estas son algunas de las que posiblemente encontrará útiles:

- **Clases:** una lista completa de todas las clases que están definidas en uno de los proyectos de su sesión, junto a todas sus funciones y variables miembros. Si pulsa sobre cualquiera de los miembros se abrirá una ventana del editor de código fuente en la posición que contiene el elemento sobre el que ha pulsado.
- **Documentos:** lista algunos de los archivos que ha usado más recientemente, ordenados según su tipo (por ejemplo archivos de código fuente, archivos de parches, documentos de texto sin formato).
- **Navegador de código:** dependiendo de la posición del cursor en un archivo, esta herramienta muestra cosas relacionadas. Por ejemplo, si el cursor está en una línea `#include`, se muestra información sobre el archivo que está incluyendo, como qué clases se declaran en él; si el cursor está en una línea en blanco que pertenece al archivo, se muestra las clases y las funciones declaradas y definidas en el archivo actual (siempre como enlaces sobre los que puede pulsar para ir al punto del archivo donde se encuentra la declaración o la definición real); si el cursor está en la definición de una función, se muestra dónde está declarada la función y se le ofrece una lista de lugares donde se usa dicha función.
- **Sistema de archivos:** le muestra una vista en forma de árbol del sistema de archivos.
- **Documentación:** le permite buscar páginas de manual y otros documentos de ayuda.
- **Fragmentos:** le proporciona secuencias de texto que puede usar una y otra vez sin tener que escribirlas en cada momento. Por ejemplo, en el proyecto del que se ha obtenido la imagen anterior, existe la necesidad de escribir frecuentemente el siguiente código

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

Se trata de una expresión poco elegante, pero siempre se mostrará del mismo modo cada vez que necesite este bucle (lo que lo convierte en un buen candidato para un fragmento de código).

- **Konsole:** abre una ventana de la línea de órdenes dentro de la ventana principal de KDevelop donde podrá introducir una orden ocasional que desee ejecutar (por ejemplo para ejecutar `./configure`).

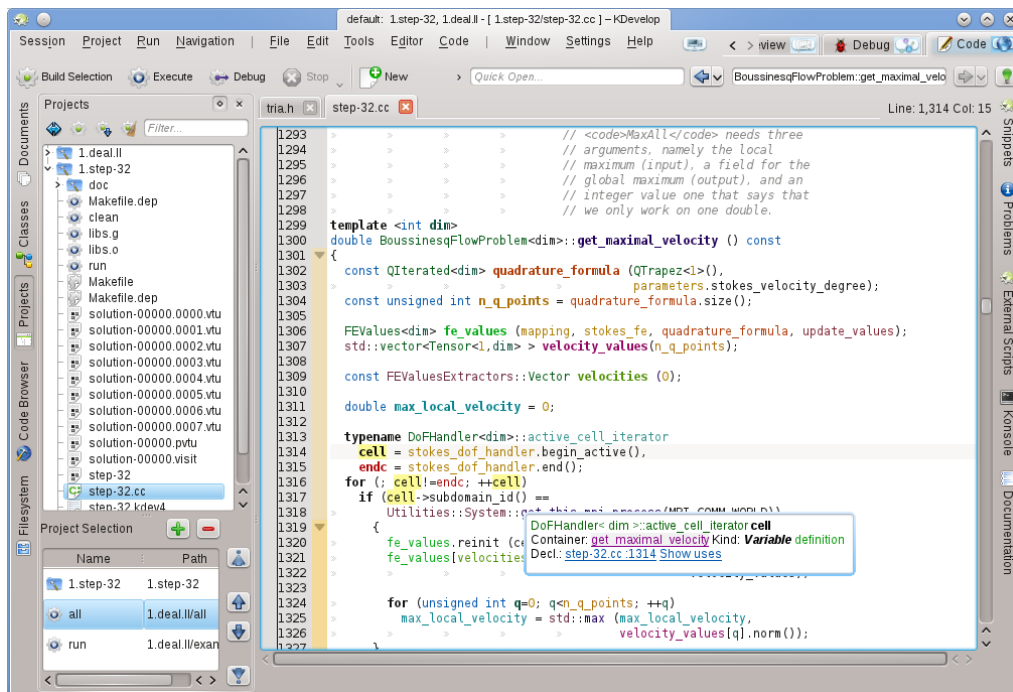
Una lista completa de herramientas y vistas se proporciona [aquí](#).

Para muchos programadores, el espacio de pantalla vertical es el más importante. Para conseguir este fin, puede ordenar las vistas de herramientas en los márgenes izquierdo y derecho de la ventana. Para mover una herramienta, pulse sobre su símbolo con el botón derecho del ratón y seleccione una nueva posición para ella.

## 3.2. Exploración del código fuente

### 3.2.1. Información local

KDevelop *entiende* el código fuente, por lo que es realmente bueno a la hora de proporcionarle información sobre las variables o funciones que aparezcan en su programa. Por ejemplo, a continuación se muestra una captura de trabajo con un trozo de código, con el cursor del ratón sobre el símbolo `cell` en la línea 1316 (si prefiere trabajar con el teclado, puede conseguir el mismo efecto manteniendo pulsada la tecla **Alt** durante unos instantes):



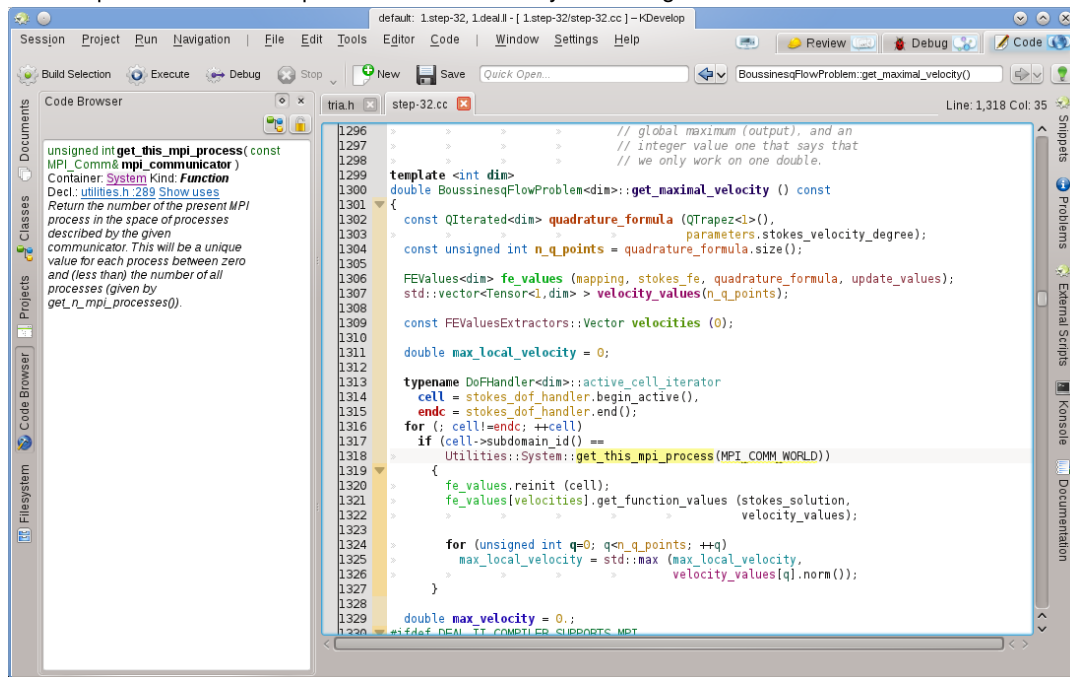
KDevelop le muestra una ayuda emergente que incluye el tipo de la variable (aquí es `DoFHandler<dim>::active_cell_iterator`), dónde se ha declarado la variable (el *contenedor*, que aquí es la función que la contiene `get_maximal_velocity`, ya que se trata de una variable local), qué es (una variable, no una función ni una clase ni un espacio de nombres) y dónde está declarada (en la línea 1314, unas cuantas de líneas más arriba en el código fuente).

En el contexto actual, el símbolo sobre el que se ha situado el cursor del ratón no posee documentación asociada. En este ejemplo, si el cursor del ratón hubiera estado sobre el símbolo `get_this_mpi_process` de la línea 1318, la información mostrada habría sido esta:



**NOTA**

La información de la ayuda emergente es fugaz: depende de que mantenga pulsada la tecla **Alt** o de la situación del cursor del ratón. Si desea un lugar permanente para ella, abra la vista de la herramienta **Navegador de código** en una de las ventanas secundarias. Por ejemplo, aquí el cursor está sobre la misma función que en el ejemplo anterior, y la vista de la herramienta de la izquierda presenta el mismo tipo de información que se mostraba en la ayuda emergente anterior:



Al mover el cursor en la parte de la derecha se modifica la información presentada en la parte de la izquierda. Aún más, si pulsa el botón **Bloquear la vista actual** que hay en la parte superior derecha podrá bloquear esta información, haciéndola independiente del movimiento del cursor mientras explora la información allí presentada.

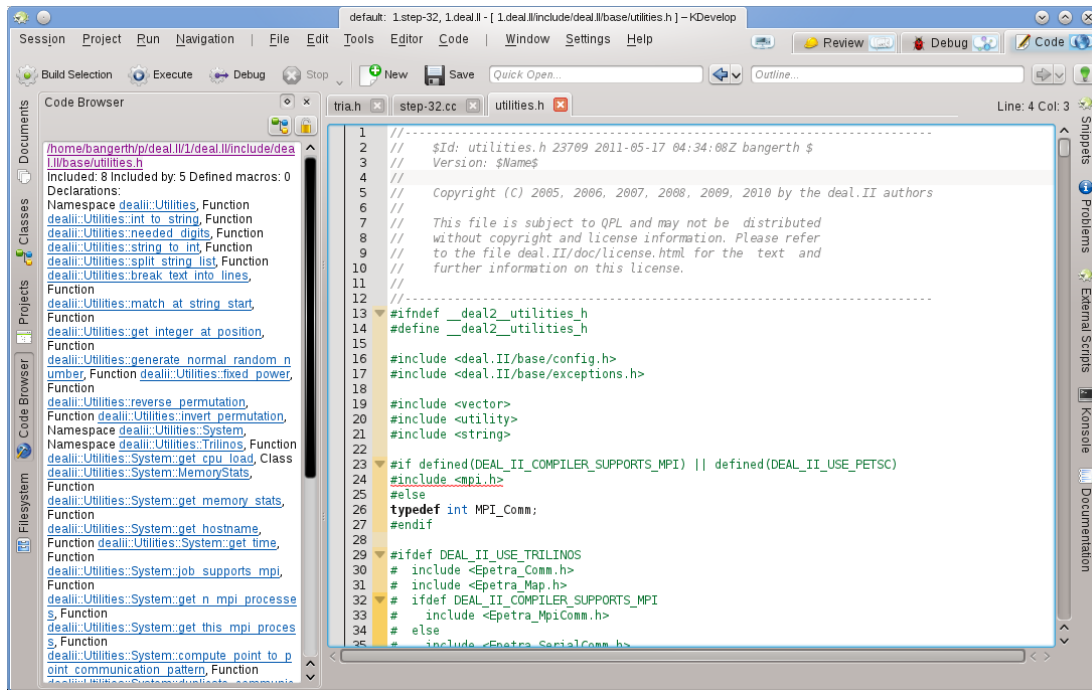
**NOTA**

Este tipo de información de contexto está disponible en muchas partes de KDevelop, no solo en el editor de código fuente. Por ejemplo, si mantiene pulsada la tecla **Alt** en una lista de completado (por ejemplo, cuando está haciendo una apertura rápida), también se muestra una información de contexto sobre el símbolo actual.

**3.2.2. Información del ámbito de los archivos**

El siguiente nivel superior es para obtener información sobre todo el archivo de código fuente en el que está trabajando actualmente. Para este fin, posicione el cursor en el ámbito del archivo en el archivo actual y mire lo que le muestra la vista de la herramienta **Navegador de código**:

## Manual de KDevelop



Aquí se muestra una lista de espacios de nombres, clases y funciones declaradas o definidas en el archivo actual, proporcionándole un resumen de lo que está ocurriendo en dicho archivo y los modos de saltar directamente a cualquiera de estas declaraciones o definiciones sin que sea necesario desplazarse arriba y abajo por el archivo o buscar un símbolo en particular.

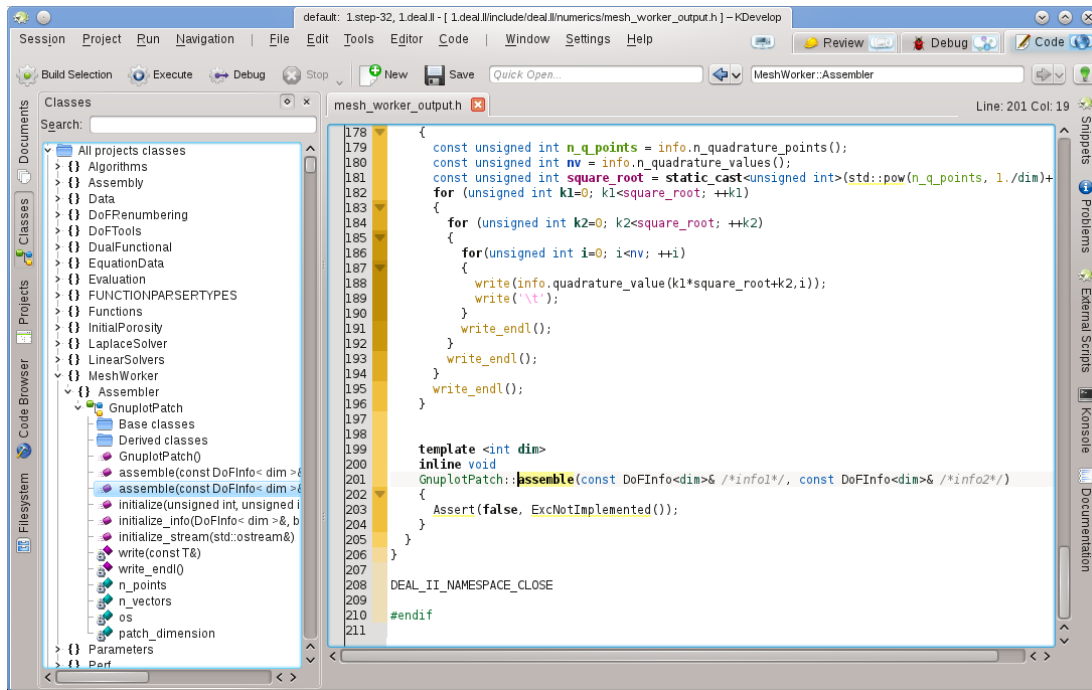
### NOTA

La información mostrada para el ámbito del archivo es la misma que se presenta en el modo 'Esquema', discutido más adelante, para navegar por el código fuente; la diferencia es que el modo de esquema es solo una ventana emergente temporal.

### 3.2.3. Información del ámbito del proyecto y de la sesión

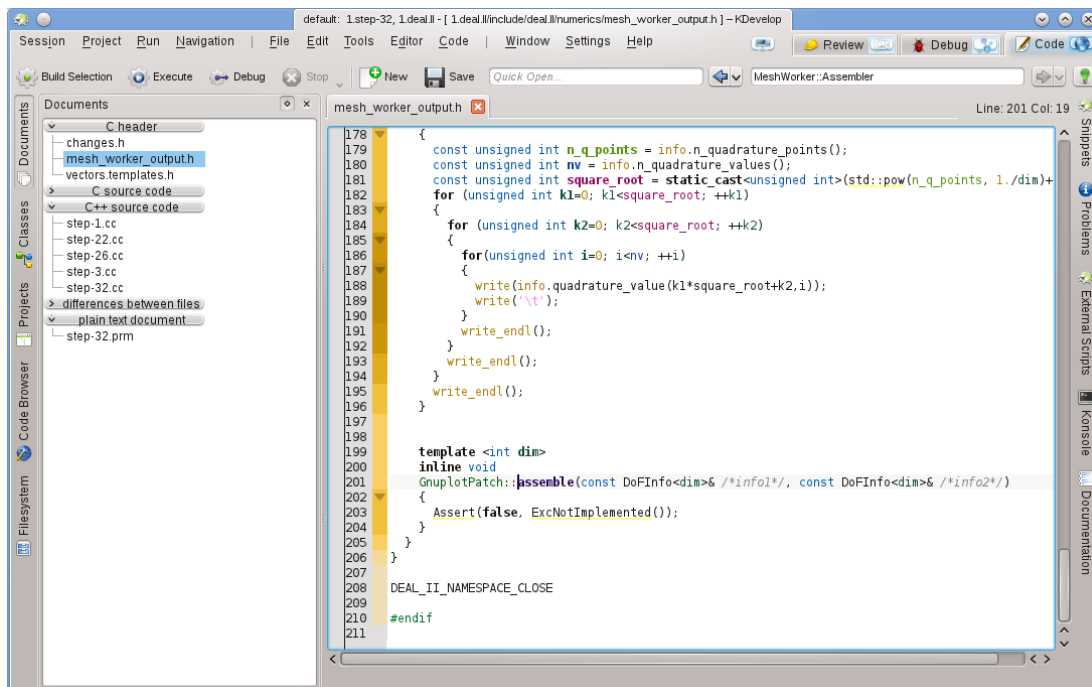
Existen diferentes maneras de obtener información sobre un proyecto completo (o, de hecho, sobre todos los proyectos de una sesión). Este tipo de información se proporciona normalmente mediante el uso de distintas vistas de herramientas. Por ejemplo, la vista de la herramienta **Clases** proporciona una estructura en forma de árbol de todas las clases y espacios de nombres circundantes de todos los proyectos de una sesión, junto a las funciones y variables miembros de cada una de dichas clases:

## Manual de KDevelop



Si sitúa el cursor sobre una entrada se le vuelve a mostrar información sobre dicho símbolo, las posiciones de su declaración y de su definición y su uso. Si hace doble clic sobre una entrada en la vista de árbol, se abrirá una ventana de edición que mostrará la posición donde se declara o define el símbolo.

Pero también existen otros modos de mostrar información global. Por ejemplo, la herramienta **Documentos** proporciona una vista de un proyecto referente a los tipos de archivos o de otros documentos que contiene el proyecto:



### 3.2.4. El resaltado con los colores del arcoíris explicado

KDevelop usa diversos colores para resaltar diferentes objetos en el código fuente. Si sabe lo que significan los distintos colores podrá obtener bastante información del código fuente con solo mirar los colores, sin necesidad de leer el código. Las reglas de resaltado son las siguientes:

- Los objetos de los tipos «class», «struct» y «enum» (los valores y el tipo), las funciones (globales) y los miembros de las clases tienen cada uno su propio color asignado: para las clases es el verde, los enumeradores se muestran en rojo oscuro y los miembros de una clase en amarillo oscuro o en morado; las funciones (globales) se muestran siempre en morado.
- Todas las variables globales se muestran en color verde oscuro.
- Los identificadores que son «typedefs» para otros tipos se muestran en color verde azulado.
- Todas las declaraciones y definiciones de objetos están en negrita.
- Si un miembro se accede dentro del contexto donde está definido (clase base o derivada) se muestra en amarillo; en caso contrario se muestra en morado.
- Si un miembro es privado o protegido, se muestra en un color algo más oscuro cuando se usa.
- Para las variables locales en el ámbito del cuerpo de una función se escogen colores del arcoíris basándose en un hash del identificador. Esto también es válido para los parámetros de la función. Un identificador siempre tendrá el mismo color dentro de su ámbito (aunque el mismo identificador tendrá distinto color si representa diferentes objetos, es decir, si se vuelve a definir en un ámbito más anidado), y normalmente también tendrá el mismo color para identificadores con el mismo nombre en distintos ámbitos. De este modo, si tiene diferentes funciones que tienen parámetros con los mismos nombres, todos los argumentos serán del mismo color. Este tipo de coloración basado en el arcoíris se puede desactivar de forma independiente de la coloración global en el diálogo de preferencias.
- Los identificadores para los que KDevelop no puede determinar la correspondiente declaración se colorean en blanco. A veces, esto se debe a la ausencia de directivas `#include`.
- Además de dicha coloración, se aplicará el resaltado de sintaxis normal del editor, como lo hace Kate. El resaltado semántico de KDevelop tendrá preferencia sobre el resaltado del editor en caso de conflicto.

## 3.3. Navegar por el código fuente

En la sección anterior hemos hablado sobre cómo explorar el código fuente, es decir, de cómo obtener información sobre símbolos, archivos y proyectos. El siguiente paso sería, pues, desplazarse por su base de código fuente, es decir, navegar por ella. De nuevo, existen diferentes niveles en los que esto es posible: local, dentro de un archivo y dentro de un proyecto.

### NOTA

Puede acceder a muchas de las formas de navegar a través del código desde el menú **Navegar** de la ventana principal de KDevelop.

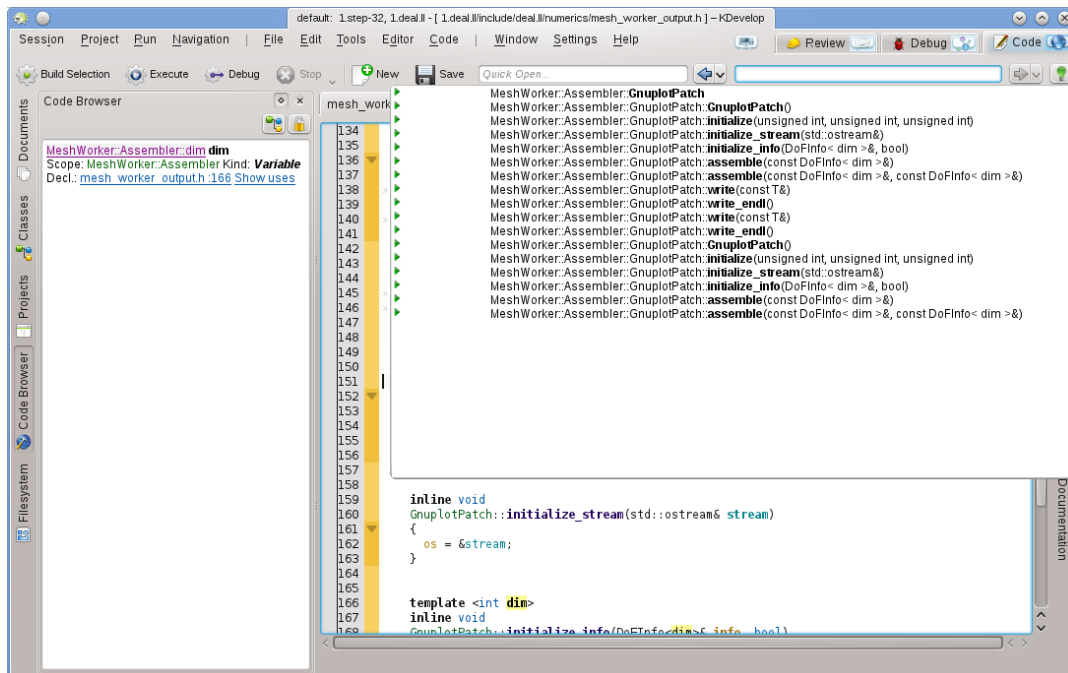
### 3.3.1. Navegación local

KDevelop es mucho más que un editor, pero *también* es un editor de código fuente. Como tal, puede mover el cursor arriba, abajo, a la izquierda y a la derecha de un archivo de código fuente, por supuesto. También puede usar las teclas **RePág** y **AvPág** y otras órdenes características de cualquier útil editor.

### 3.3.2. Navegación en el ámbito de archivos y modo de esquema

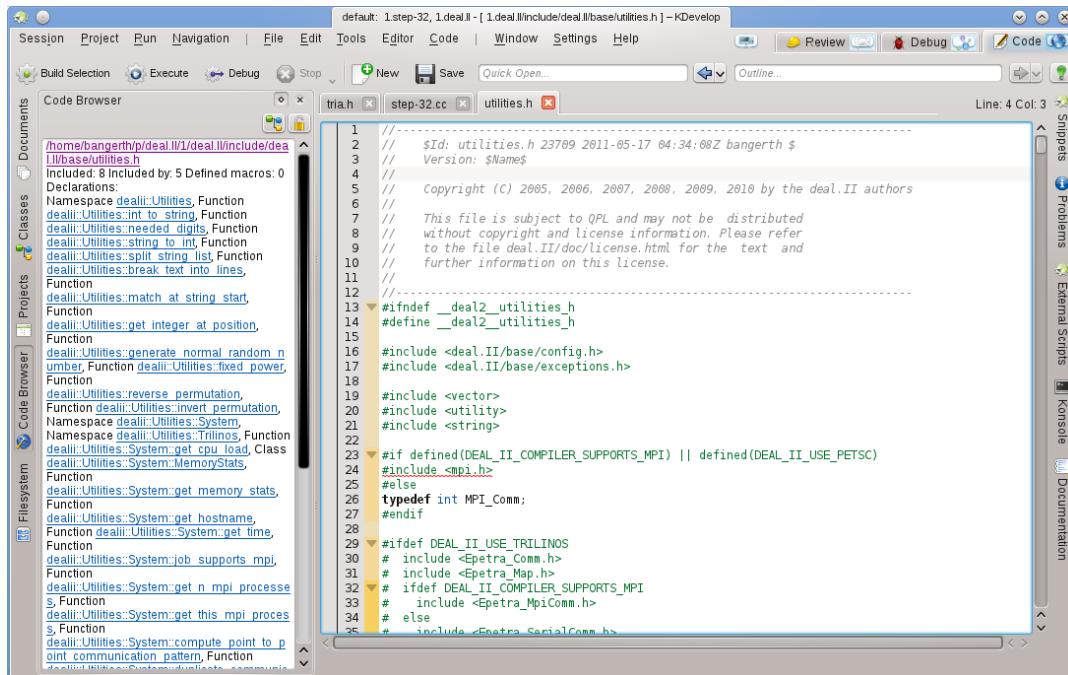
En el ámbito de archivos, KDevelop ofrece diversas posibilidades de navegar a través del código fuente. Por ejemplo:

- **Esquema:** Puede obtener un esquema de lo que hay en el archivo actual de al menos tres modos diferentes:
  - Si pulsa en el campo de texto **Esquema** que hay en la parte superior derecha de la ventana principal o si pulsa **Alt-Ctrl-N**, se abre un menú desplegable que lista todas las declaraciones de funciones y de clases:



A continuación puede seleccionar a cual de ellas saltar o (si hay muchas) comenzar a teclear cualquier texto que deba aparecer en los nombres que se muestran; en este caso, a medida que va tecleando, la lista se va haciendo cada vez más pequeña para eliminar los nombres que no coincidan con el texto que está escribiendo hasta que pueda seleccionar una de las entradas.

- Si posiciona el cursor en el ámbito de un archivo (es decir fuera de cualquier declaración o definición de función o clase) mientras tiene abierta la herramienta **Navegador de código**:



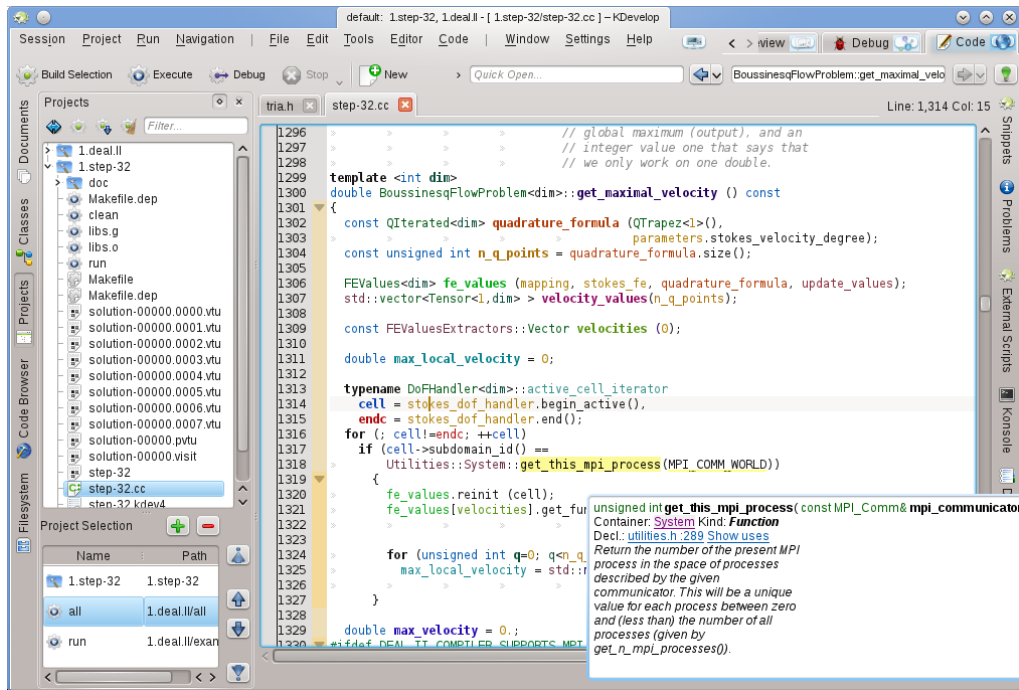
Esto también le proporciona un esquema de lo que está ocurriendo en el archivo actual, y le permite seleccionar el lugar al que desea saltar.

- Si sitúa el puntero del ratón sobre la pestaña de uno de los archivos abiertos también se muestra un esquema del archivo en dicha pestaña.
- Los archivos de código fuente se organizan como una lista de declaraciones o definiciones de funciones. Si pulsa **Alt-Ctrl-RePág** y **Alt-Ctrl-AvPág** podrá saltar a la definición de la anterior o siguiente función de este archivo.

### 3.3.3. Navegación en el ámbito del proyecto y de la sesión: navegación semántica

Como se ha mencionado en otras partes, KDevelop no suele considerar los archivos de código fuente de forma individual, sino que contempla los proyectos como un todo (o, mejor, todos los proyectos que forman parte de la sesión actual). Como consecuencia, ofrece muchas posibilidades para navegar a través de todos los proyectos. Algunas de ellas derivan de lo que ya se ha discutido en la sección sobre [Exploración del código fuente](#), mientras que otras son realmente diferentes. El punto en común reside en que estas funcionalidades de navegación están basadas en un *entendimiento semántico* del código fuente (es decir, le ofrecen algo que necesita analizar proyectos completos y conectar datos). La lista siguiente muestra algunos modos de navegación a través del código fuente que puede estar distribuido en un gran número de archivos:

- Como se ha visto en la sección sobre [Exploración del código fuente](#), puede obtener una ayuda emergente con información sobre espacios de nombres, clases, funciones o variables individuales con solo situar el cursor del ratón sobre dichos elementos o manteniendo pulsada la tecla **Alt** durante un rato. Aquí tiene un ejemplo:



Si pulsa sobre los enlaces de la declaración de un símbolo o expande la lista de usos podrá saltar a dichas posiciones, abriendo su respectivo archivo si es necesario y situando el cursor en la correspondiente posición. Puede conseguir un efecto similar usando la herramienta del **Navegador de código**, como se ha discutido anteriormente.

- Un modo más rápido de ir a la declaración de un símbolo sin necesidad de pulsar los enlaces de la ayuda emergente consiste en activar temporalmente el **modo de exploración del código** manteniendo pulsada la tecla **Alt** o la tecla **Ctrl**. En este modo es posible pulsar directamente sobre cualquier símbolo del editor para ir a su declaración.
- **Apertura rápida**: un modo muy potente de saltar a otros archivos o posiciones consiste en usar los diversos métodos de *apertura rápida* de KDevelop. Existen cuatro versiones de ellos:
  - **Apertura rápida de clase** (**Navegar** → **Apertura rápida de clase** o **Alt-Ctrl-C**): obtendrá una lista de todas las clases de la sesión actual. Comience a teclear el nombre de la clase y la lista le mostrará solo las que realmente se ajusten a lo que vaya escribiendo. Si la lista es lo suficientemente corta, seleccione un elemento usando las teclas arriba y abajo para que KDevelop le lleve al lugar donde se haya declarado la clase.
  - **Apertura rápida de función** (**Navegar** → **Apertura rápida de función** o **Alt-Ctrl-M**): obtendrá una lista de todas las funciones (miembro) que forman parte de los proyectos de la sesión actual, donde podrá seleccionar un elemento del mismo modo que se ha descrito anteriormente. Tenga en cuenta que esta lista puede incluir tanto declaraciones como definiciones de funciones.
  - **Apertura rápida de archivo** (**Navegar** → **Apertura rápida de archivo** o **Alt-Ctrl-O**): obtendrá una lista de todos los archivos que forman parte de los proyectos de la sesión actual, donde podrá seleccionar un elemento del mismo modo que se ha descrito anteriormente.
  - **Apertura rápida universal** (**Navegar** → **Apertura rápida** o **Alt-Ctrl-Q**): si olvida la combinación de teclas que está enlazada a las acciones anteriores, esta es la navaja suiza universal que necesita (le muestra una lista combinada de todos los archivos, funciones, clases y otras cosas de entre las que puede seleccionar una de ellas).
- **Saltar a la declaración/definición**: cuando se está implementando una función (miembro), a menudo es necesario volver al punto donde se ha declarado la función, por ejemplo para mantener la lista de los argumentos de la función sincronizada entre la declaración y la definición, o para actualizar la documentación. Para ello, sitúe el cursor sobre el nombre de la función

y seleccione **Navegar** → **Saltar a la declaración** (o pulse **Ctrl-.**) para ir al lugar donde se ha declarado la función. Existen diversos modos de volver al lugar original:

- Seleccionando **Navegar** → **Saltar a definición** (o pulsando **Ctrl-.**).
- Seleccionando **Navegar** → **Anterior contexto visitado** (o pulsando **Meta-Izquierda**), como se describe a continuación.

#### NOTA

El salto a la declaración de un símbolo no solo funciona cuando sitúa el cursor sobre el nombre de la función que esté implementando en un momento dado. Además, también funciona con otros símbolos: si sitúa el cursor sobre una variable (local, global o miembro) y salta a su declaración, también lo llevará al lugar donde se declara. De modo similar, puede situar el cursor sobre el nombre de una clase (por ejemplo, en una variable de declaración de función) y saltar al lugar donde se declara.

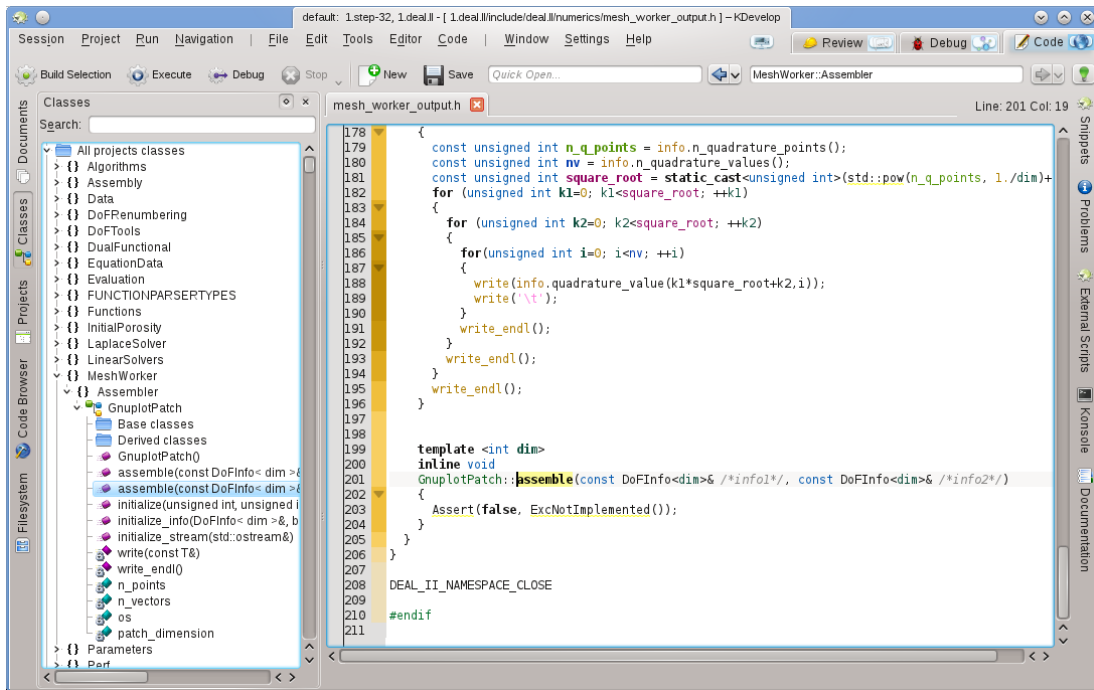
- **Cambiar entre declaración/definición:** en el ejemplo anterior, para ir a la lugar de la declaración de la función actual, necesita primero colocar el cursor sobre el nombre de la función. Para evitar este paso, puede seleccionar **Navegación** → **Cambiar entre definición/declaración** (o pulse **Mayúsculas-Ctrl-C**) para ir a la declaración de la función dentro de la que está el cursor en ese momento. Si selecciona la misma entrada de menú por segunda vez, volverá al lugar donde se define la función.
- **Uso anterior/siguiente:** si sitúa el cursor sobre el nombre de una variable local y selecciona **Navegación** → **Siguiente uso** (o pulsa **Meta-Mayúsculas-Derecha**) irá al siguiente uso de dicha variable en el código. (Tenga en cuenta que esto no busca solo la siguiente coincidencia del nombre de la variable, sino que también tiene en cuenta que las variables con el mismo nombre que estén en diferentes ámbitos son distintas). Esto mismo también funciona con los nombres de las funciones. Si selecciona **Navegación** → **Anterior uso** (o pulsa **Meta-Mayúsculas-Izquierda**) irá al anterior uso de un símbolo.

#### NOTA

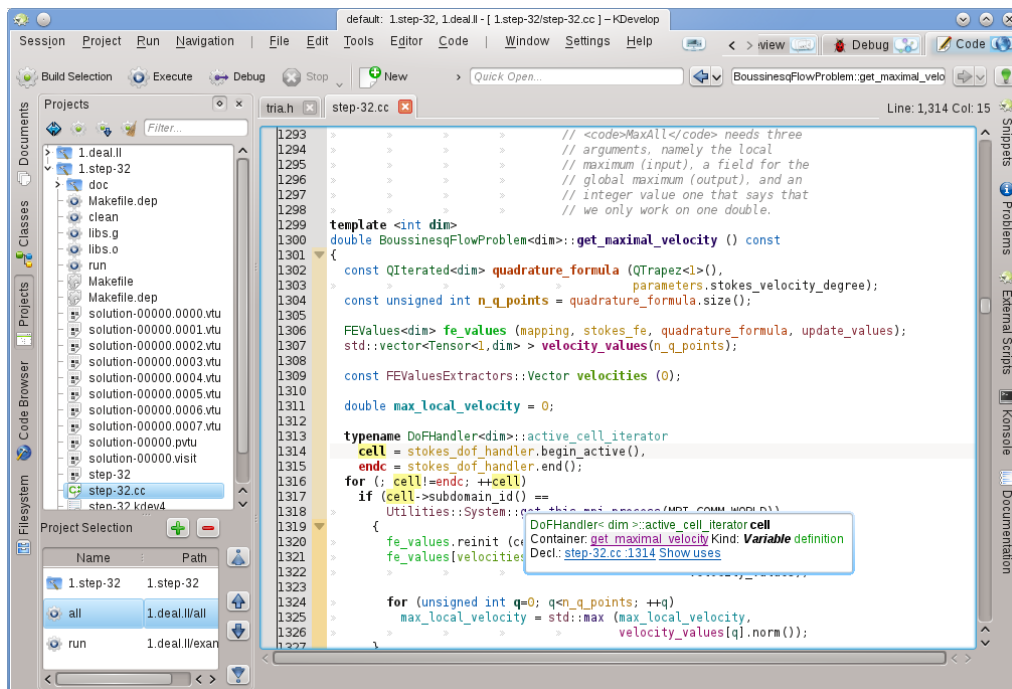
Para ver una lista con todos los usos de un nombre sobre los que funcionan estas órdenes, sitúe el cursor sobre él y abra el visor de la herramienta **Navegador de código**, o bien mantenga pulsada la tecla **Alt**. Esto se explica con mayor detalle en la sección sobre [exploración del código](#).

- **La lista de contexto:** los navegadores web tienen esta función con la que se puede ir atrás y adelante en la lista de las páginas web recientemente visitadas. KDevelop posee este mismo tipo de funcionalidad, excepto que, en lugar de páginas web, visita *contextos*. Un contexto es la posición actual del cursor, que puede cambiar navegando con cualquier método excepto órdenes de cursor (por ejemplo, pulsando en una dirección proporcionada por una ayuda emergente, en la vista de la herramienta **Navegador de código**, cualquiera de las opciones del menú **Navegación** o cualquier otra orden de navegación. El uso de **Navegación** → **Contexto visitado anteriormente** (**Meta-Izquierda**) y **Navegación** → **Siguiente contexto visitado** (**Meta-Derecha**) le lleva de un lado a otro de la lista de contextos visitados, del mismo modo que los botones **atrás** y **adelante** de un navegador le llevan a la página web anterior o siguiente de la lista de páginas visitadas.
- Finalmente, hay vistas de herramientas que le permiten navegar a diferentes lugares de su base de código fuente. Por ejemplo, la herramienta **Clases** le proporciona una lista de todos los espacios de nombres y clases de todos los proyectos de la sesión actual, y le permite desplegar sus elementos para ver las funciones y variables miembros de cada una de las clases:

## Manual de KDevelop



Si hace doble clic sobre un elemento (o si usa la correspondiente opción del menú de contexto que se muestra al pulsar el botón derecho del ratón) puede saltar a la posición de la declaración de dicho elemento. Otras herramientas permiten hacer cosas similares; por ejemplo, la herramienta **Proyectos** proporciona una lista de los archivos que forman parte de una sesión:



De nuevo, si hace doble clic sobre un archivo, se abrirá.

## 3.4. Escritura de código fuente

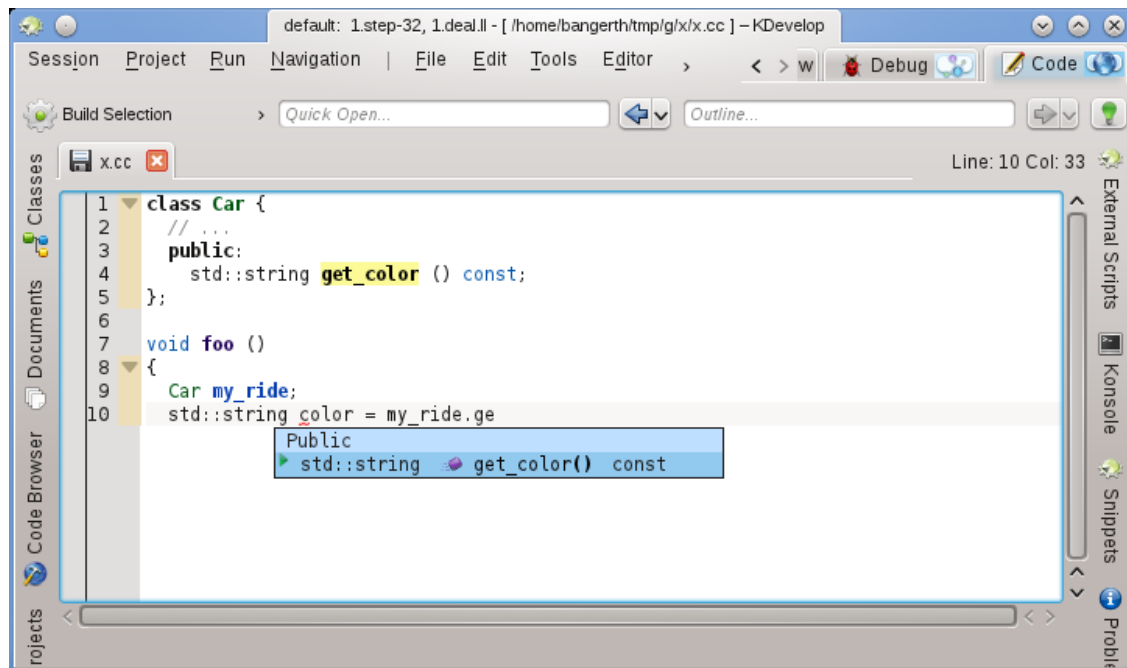
Debido a que KDevelop entiende el código fuente de sus proyectos, le puede ayudar al escribir más código fuente. Lo siguiente ilustra algunas de las formas en las que hace esto.

### 3.4.1. Terminación automática

Probablemente, una de las características más útiles cuando se escribe código es la terminación automática. Considere, por ejemplo, el siguiente código:

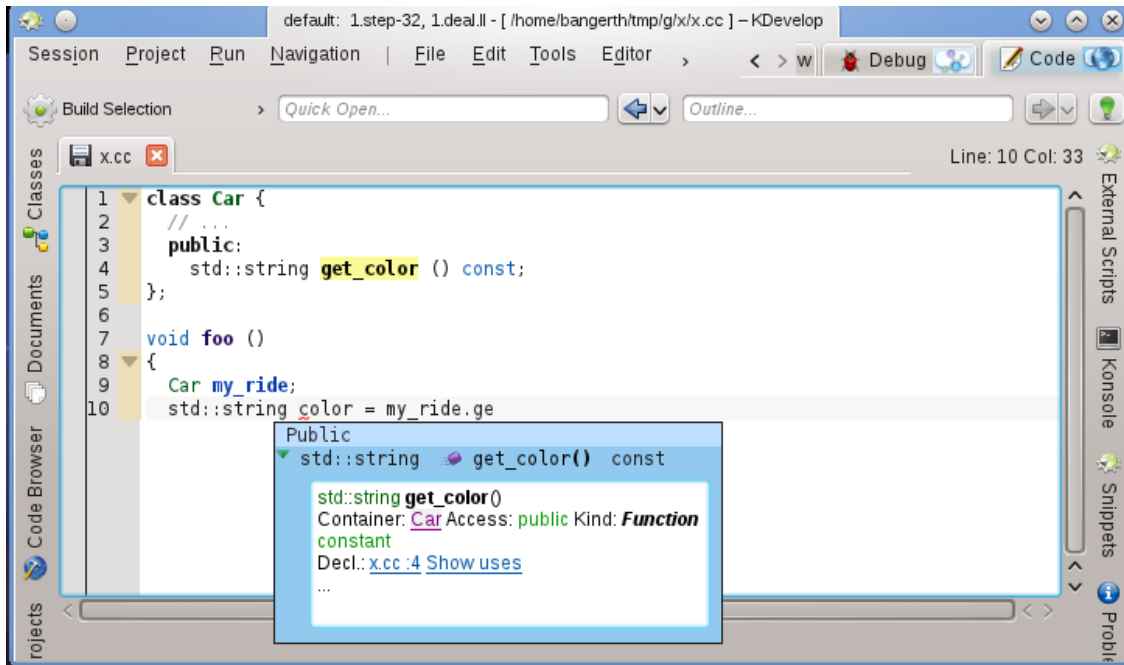
```
class Car {
    // &#8230;
    public:
        std::string get_color () const;
};
void foo()
{
    Car my_ride;
    // &#8230; hacer algo en esta variable&#8230;
    std::string color = my_ride.ge
```

En la última línea, KDevelop recordará que la variable `my_ride` es de tipo `Car`, y ofrecerá completar de forma automática el nombre de la función miembro `ge` como `get_color`. De hecho, todo lo que tiene que hacer es seguir tecleando hasta que la función de terminación automática haya reducido el número de coincidencias a una, y luego pulsar la tecla **Intro**:



Tenga en cuenta que puede hacer clic sobre la ayuda emergente para obtener más información sobre la función, además de su tipo de valor devuelto y de si es pública o no:

## Manual de KDevelop



La terminación automática puede ahorrarle bastantes pulsaciones de teclas si su proyecto usa nombres largos para funciones y variables; además, evita que escriba mal estos nombres (y los consiguientes errores del compilador) y hace que sea más fácil recordar los nombres exactos de las funciones; por ejemplo, si todos sus «getters» comienzan por `get_`, la función de terminación automática solo le mostrará la lista de los posibles «getters» en cuanto haya tecleado las cuatro primeras letras, probablemente recordándole durante el proceso cuál de las funciones es la correcta. Tenga en cuenta que para que funcione la terminación automática, no hace falta que la declaración de la clase `Car` ni la variable `my_ride` estén en el mismo archivo donde esté escribiendo código. KDevelop tiene que saber que dichas clases y variables están conectadas; es decir, los archivos en los que se realizan estas conexiones tienen que ser parte del proyecto en el que esté trabajando.

### NOTA

KDevelop no siempre sabe cuándo debe ayudarle a completar código. Si la ayuda emergente de terminación automática no se muestra, pulse **Ctrl-Espacio** para abrir de forma manual la lista de terminaciones. En general, para que funcione la terminación automática, KDevelop necesita analizar sus archivos de código fuente. Esto se realiza en segundo plano con todos los archivos que forman parte de los proyectos de la sesión actual en cuanto inicia KDevelop, así como cuando deja de teclear durante una fracción de segundo (esta pausa se puede configurar).

### NOTA

KDevelop analiza únicamente los archivos que considera de código fuente, según se determina por el tipo MIME del archivo. Este tipo no se fija antes de la primera vez que se guarda un archivo; por lo tanto, cuando cree un nuevo archivo y comience a escribir código en él, no se activará el análisis para la terminación automática hasta que lo haya guardado por primera vez.

**NOTA**

Al igual que en la nota anterior, para que funcione la terminación automática, KDevelop debe ser capaz de encontrar declaraciones en los archivos de cabecera. Para ello, busca en cierto número de rutas por omisión. Si no encuentra automáticamente un archivo de cabecera, subrayará en rojo su nombre; en este caso, pulse con el botón derecho del ratón sobre él para indicarle a KDevelop de forma explícita dónde puede encontrar dicho archivo y qué información proporciona.

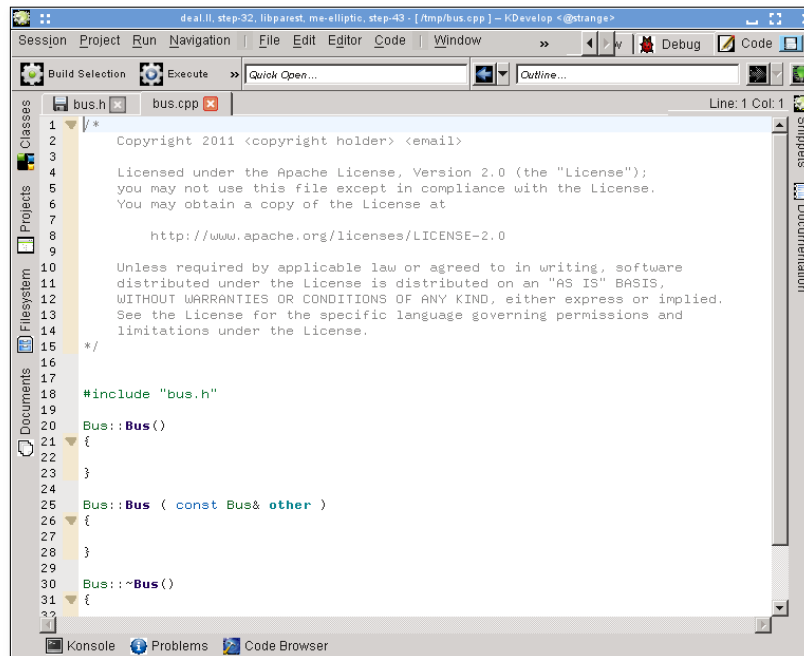
**NOTA**

La configuración de la terminación automática se describe en [esta sección de este manual](#).

### 3.4.2. Añadir nuevas clases e implementar funciones miembro

KDevelop tiene un asistente para añadir nuevas clases. El procedimiento se describe en [Creación de una nueva clase](#). Se puede crear una clase sencilla de C++ escogiendo la plantilla de C++ básico en la categoría Clase. En el asistente podemos escoger algunas funciones miembro predefinidas, como, por ejemplo: un constructor vacío, un constructor de copia o un destructor.

Tras completar el asistente, se crean los nuevos archivos y se abren en el editor. El archivo de cabecera ya contiene salvaguardas de inclusión y la nueva clase posee todas las funciones miembro que hayamos seleccionado. Los dos siguientes pasos deberían ser documentar la clase y sus funciones miembro, y luego implementarlas. Más adelante hablaremos sobre cómo documentar clases y funciones. Para implementar las funciones especiales que ya hemos añadido, iremos la pestaña **bus.cpp**, donde ya se ha proporcionado un esqueleto para las funciones.



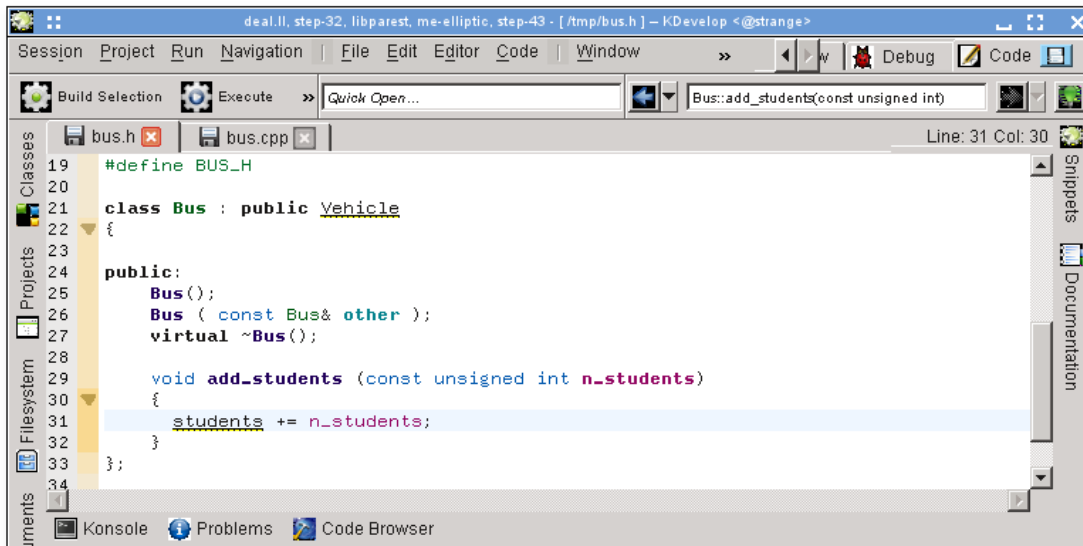
```

1  /*
2  Copyright 2011 <copyright holder> <email>
3
4  Licensed under the Apache License, Version 2.0 (the "License");
5  you may not use this file except in compliance with the License.
6  You may obtain a copy of the License at
7
8  http://www.apache.org/licenses/LICENSE-2.0
9
10 Unless required by applicable law or agreed to in writing, software
11 distributed under the License is distributed on an "AS IS" BASIS,
12 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 See the License for the specific language governing permissions and
14 limitations under the License.
15 */
16
17
18 #include "bus.h"
19
20 Bus::Bus()
21 {
22 }
23
24 Bus::Bus ( const Bus& other )
25 {
26 }
27
28
29 Bus::~Bus()
30 {
31 }
32

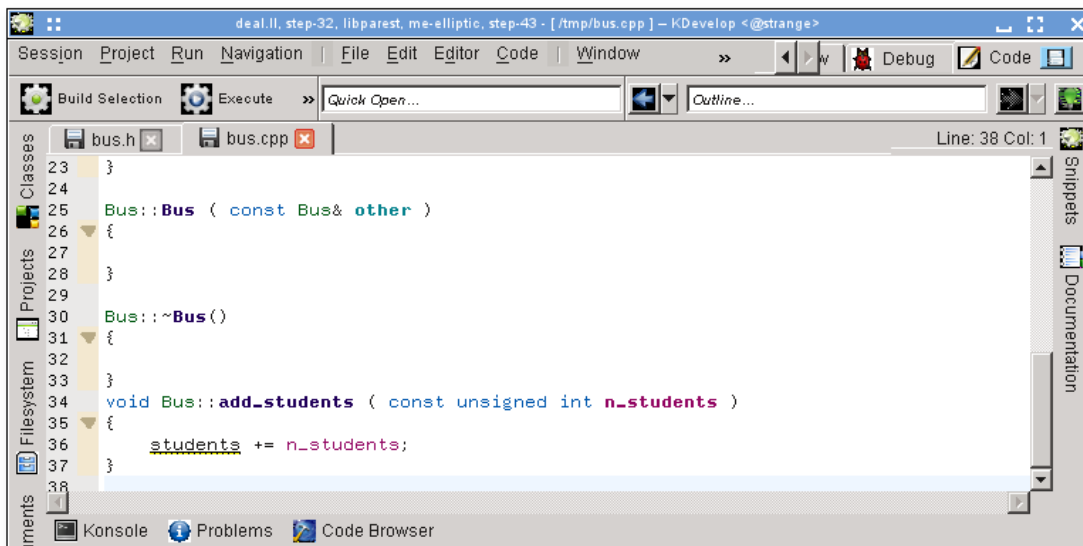
```

Para añadir nuevas funciones miembro, vuelva a la pestaña **bus.h** y añada el nombre de una función. Por ejemplo, añadamos esta:

## Manual de KDevelop

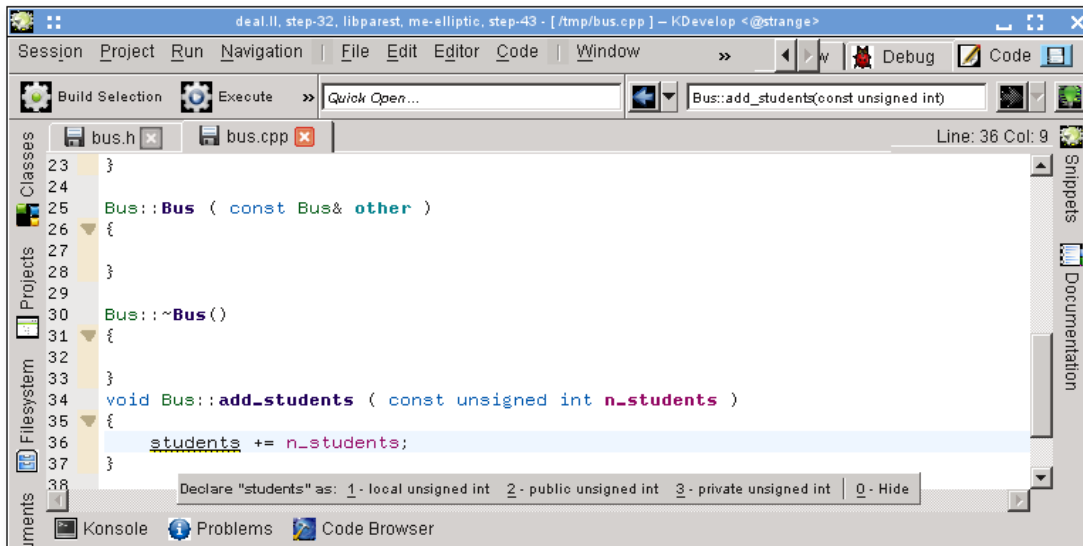


Tenga en cuenta cómo hemos empezado ya con la implementación. No obstante, en muchos estilos de programación, la función no se debe implementar en el archivo de cabecera sino en el correspondiente archivo `.cpp`. Para ello, sitúe el cursor sobre el nombre de la función y seleccione **Código** → **Mover al código fuente** o pulse **Ctrl-Alt-S**. Esto eliminará del archivo de cabecera el código que hay entre las llaves (sustituyéndolo por un punto y coma como es debido para terminar la declaración de la función) y lo moverá al archivo de código fuente:

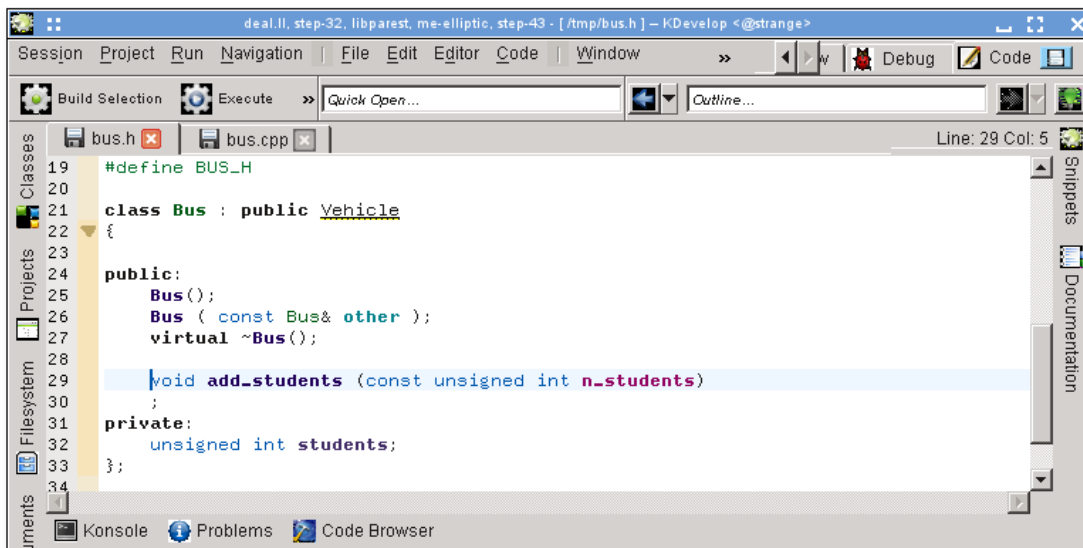


Tenga en cuenta cómo hemos comenzado a escribir y que queremos insinuar que la variable `students` debe ser probablemente miembro de la clase `Bus`, aunque aún no la hemos añadido. Note también cómo la subraya KDevelop para dejar claro que no sabe nada sobre la variable. Pero este problema se puede resolver: si pulsa sobre el nombre de la variable se muestra la siguiente ayuda emergente:

## Manual de KDevelop

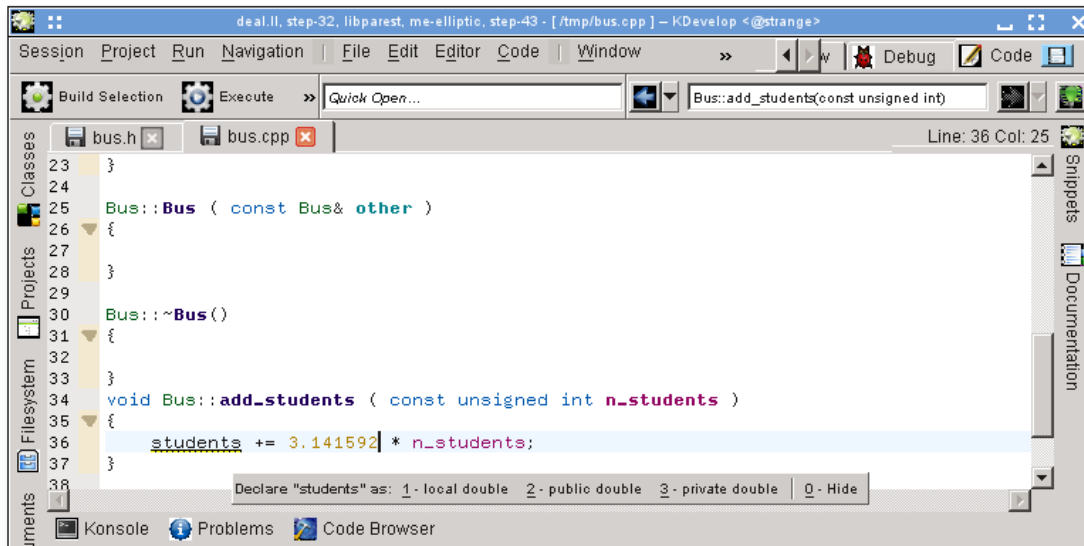


(Se puede conseguir lo mismo haciendo clic sobre ella con el botón derecho y seleccionando **Resolver: Declarar como**). Seleccionemos '3 - private unsigned int' (con el ratón o pulsando **Alt-3**) y veamos cómo se muestra en el archivo de cabecera:

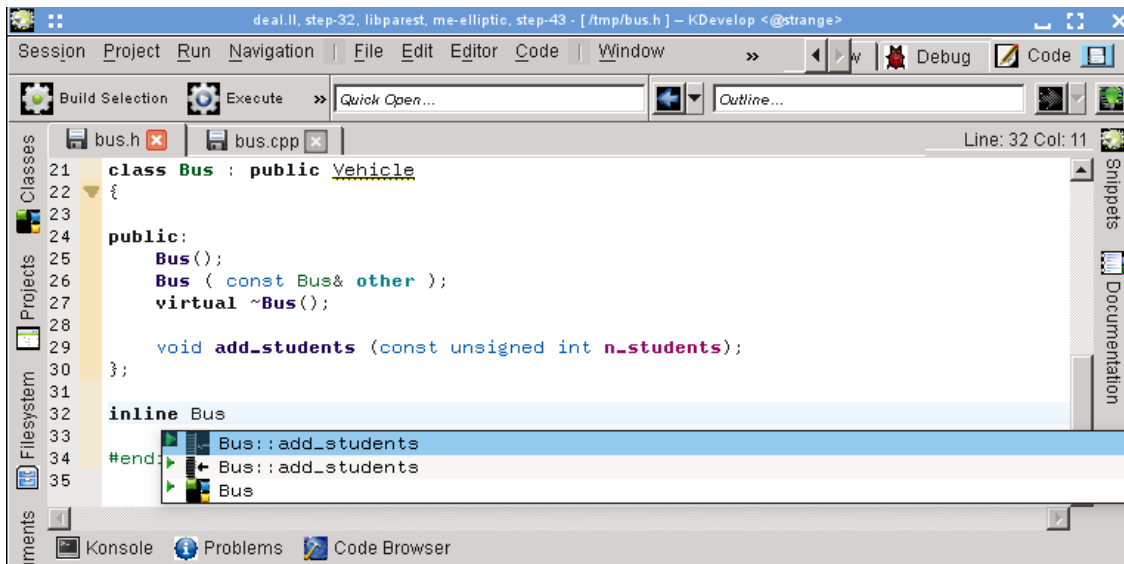


Vale la pena tener en cuenta que KDevelop extrae el tipo de la variable a declarar de la expresión usada para inicializarla. Por ejemplo, si hemos escrito la adición del siguiente modo ambiguo, habría sugerido declarar la variable de tipo `double`:

## Manual de KDevelop

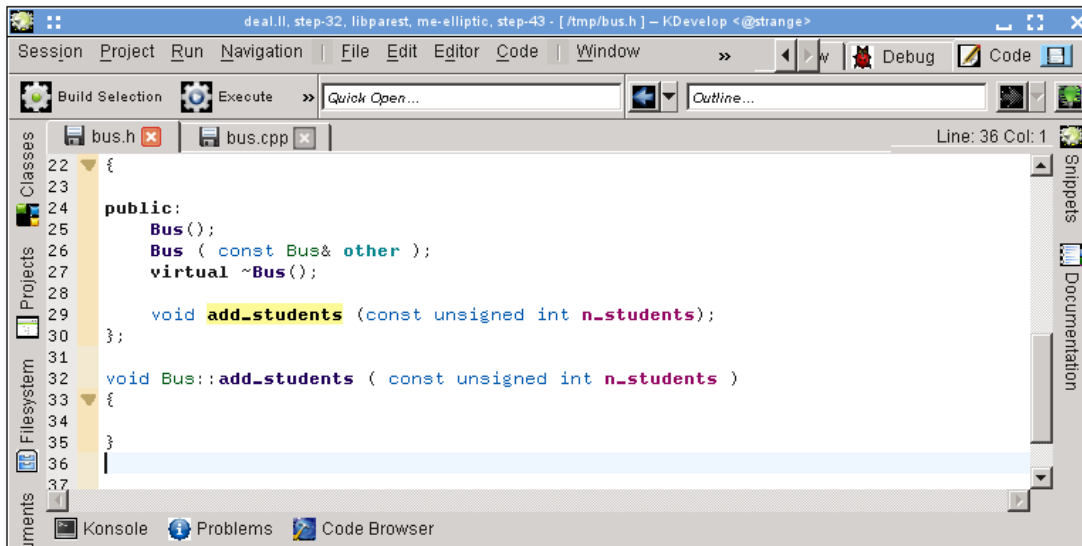


Como observación final, el método que usa **Código** → **Mover al código fuente** no siempre inserta la nueva función miembro donde usted desearía. Por ejemplo, es posible que quisiera marcarla como `inline` y situarla al final del archivo de cabecera. En este caso, escriba la declaración al principio escribiendo la definición de la función como sigue:



KDevelop ofrece de forma automática todas las posibles terminaciones de lo que debe aparecer aquí. Si selecciona una de las dos entradas `add_students`, se muestra el siguiente código que ya rellena la lista de argumentos completa:

## Manual de KDevelop



### NOTA

En el ejemplo, si se acepta una de las opciones, la herramienta de terminación automática ofrece la firma correcta; pero, por desgracia, borra el marcador `inline` que ya estaba escrito. Se ha informado de este problema como [Bug 274245 de KDevelop](#).

### 3.4.3. Documentar declaraciones

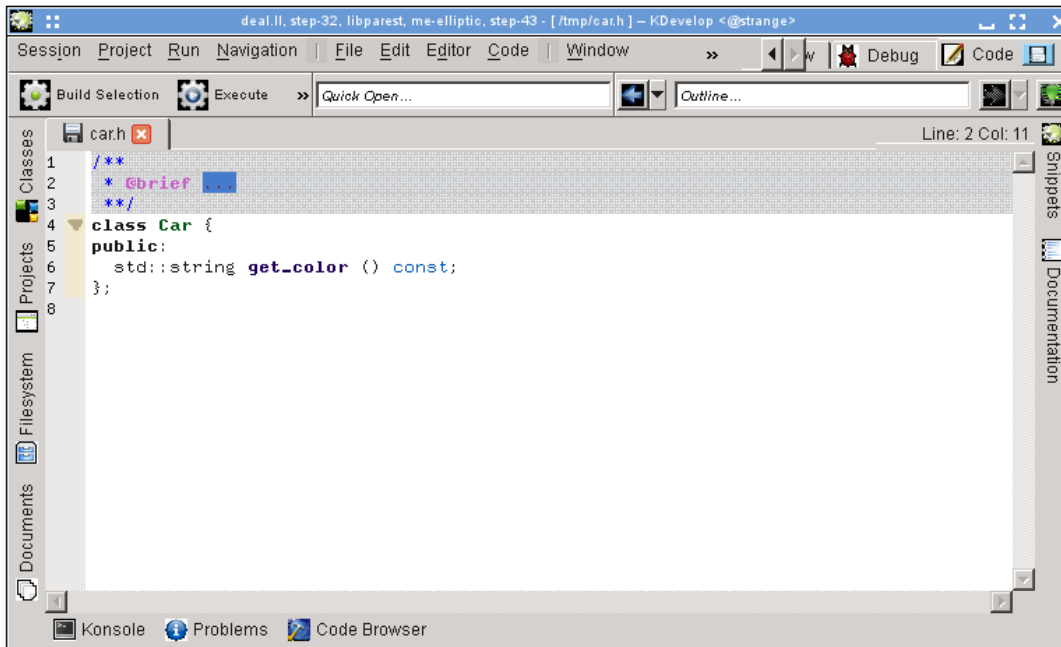
El buen código está bien documentado, tanto a nivel de la implementación de algoritmos dentro de las funciones como a nivel de la interfaz — es decir, las clases, las funciones (miembro y globales) y las variables (miembro y globales) deben estar documentadas para explicar su propósito, los posibles valores de los argumentos, las condiciones previas y posteriores, etc. Por lo que a la documentación de la interfaz se refiere, [doxygen](#) se ha convertido en el estándar de facto para formatear comentarios que luego se pueden extraer y mostrar en páginas web en las que se pueden realizar búsquedas.

KDevelop permite usar este estilo de comentarios mediante un atajo de teclado para generar la infraestructura de los comentarios que documentan una clase o una función miembro. Por ejemplo, suponiendo que ya ha escrito el siguiente código:

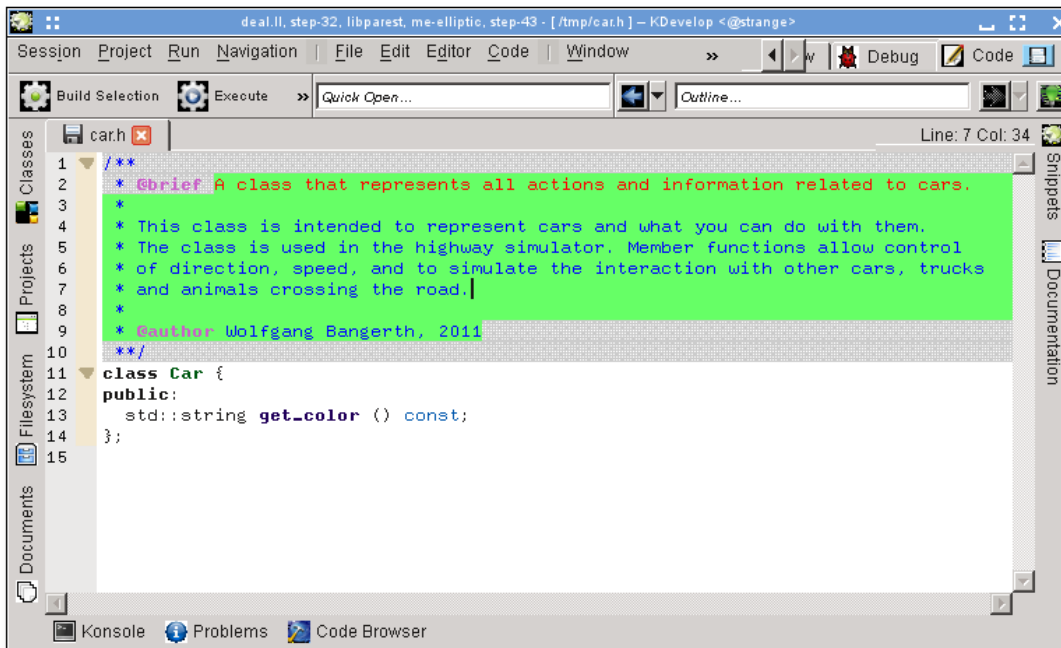
```
class Car {
public:
    std::string get_color () const;
};
```

Ahora desea añadir documentación tanto a la clase como a la función miembro. Para ello, mueva el cursor a la primera línea y seleccione **Código** → **Documentar declaración** o pulse **Alt-Mayúsculas-D**. KDevelop le responderá con lo siguiente:

## Manual de KDevelop



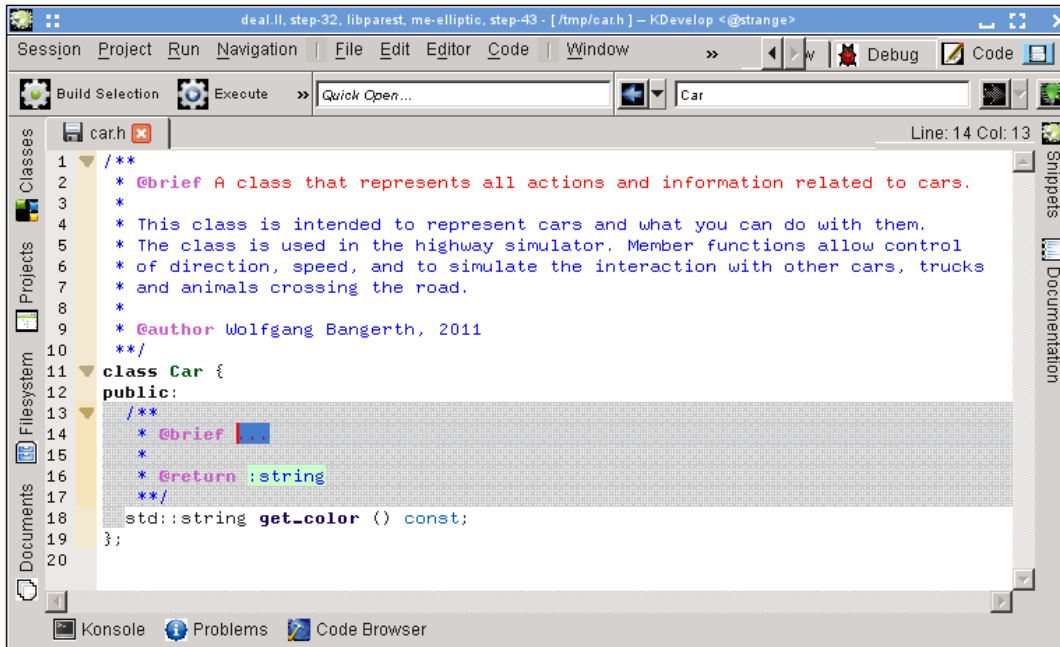
El cursor ya está en el área de color gris para que pueda rellenar la descripción corta (tras la palabra clave de doxygen `@brief`) de esta clase. Después puede continuar añadiendo documentación a este comentario que dé una descripción más detallada de lo que hace la clase:



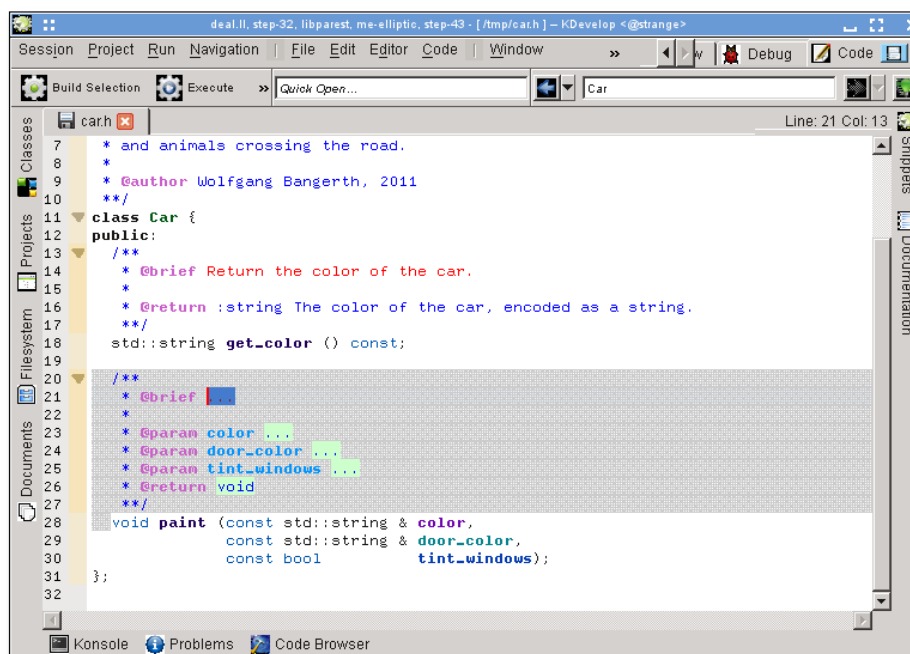
Mientras el editor esté dentro del comentario, el texto comentado se resalta en verde (y volverá a su color normal en cuanto mueva el cursor fuera del comentario). Cuando llegue al final de una línea, pulse **Intro** para que KDevelop cree de forma automática una nueva línea que comience con un asterisco y sitúe el cursor dos caracteres más a la derecha.

## Manual de KDevelop

Ahora vamos a comentar la función miembro, situando de nuevo el cursor sobre la línea de la declaración y seleccionando **Código** → **Documentar declaración** o pulsando **Alt-Mayúsculas-D**:



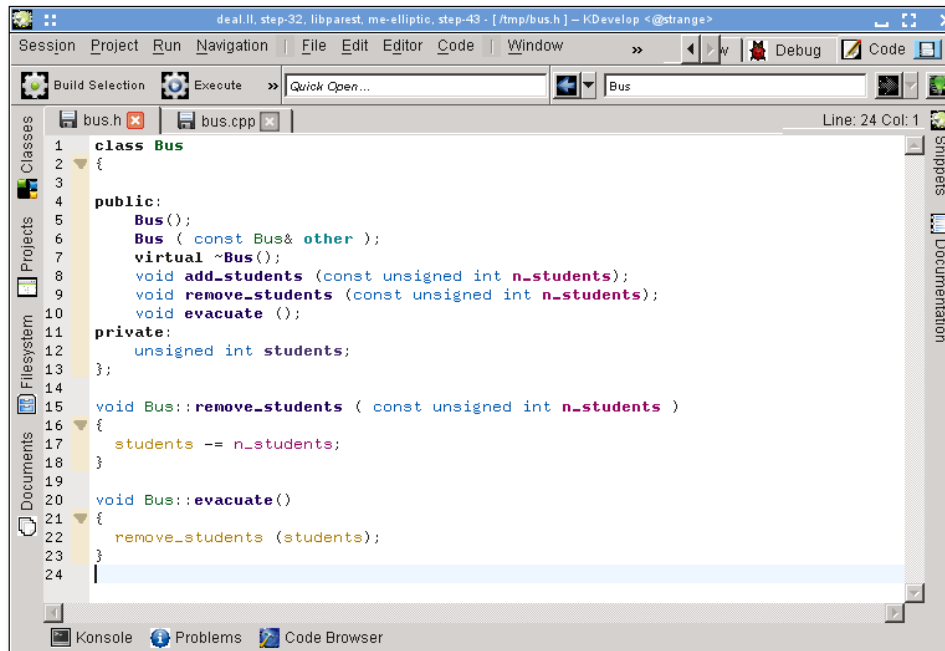
De nuevo, KDevelop genera automáticamente el esqueleto del comentario, incluyendo la documentación para la función, así como el tipo de dato que devuelve. En el caso actual, el nombre de la función es bastante autoexplicativo, pero a veces los argumentos de la función pueden no estar presentes y se deben comentar de forma individual. Para ilustrar este punto, consideremos una función un poco más interesante y el comentario que KDevelop genera de forma automática:



Aquí, el comentario sugerido ya contiene todos los campos de Doxygen para los distintos parámetros, por ejemplo.

### 3.4.4. Cambiar el nombre de variables, funciones y clases

A veces es necesario cambiar el nombre de una función, clase o variable. Por ejemplo, supongamos que ya tenemos esto:



```

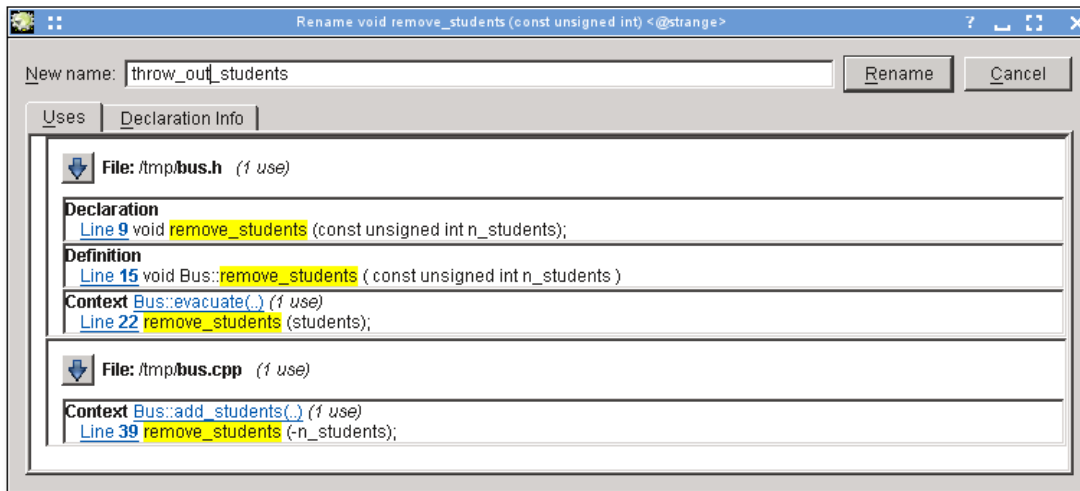
1  class Bus
2  {
3
4  public:
5      Bus();
6      Bus ( const Bus& other );
7      virtual ~Bus();
8      void add_students (const unsigned int n_students);
9      void remove_students (const unsigned int n_students);
10     void evacuate ();
11 private:
12     unsigned int students;
13 };
14
15 void Bus::remove_students ( const unsigned int n_students )
16 {
17     students -= n_students;
18 }
19
20 void Bus::evacuate()
21 {
22     remove_students (students);
23 }
24

```

Entonces caemos en la cuenta de que no nos gusta el nombre `remove_students` y que hubiera sido mejor llamarla, por ejemplo, `throw_out_students`. Podríamos realizar una búsqueda con sustitución del nombre, pero eso conllevaría un par de problemas:

- La función se puede usar en más de un archivo.
- Lo único que realmente queremos hacer es cambiar el nombre de esta función y no tocar otras funciones que pudieran tener el mismo nombre y que estuvieran declaradas en otras clases o espacios de nombres.

Ambos problemas se pueden solucionar situando el cursor sobre cualquiera de las apariciones del nombre de la función y seleccionando **Código** → **Cambiar nombre de la declaración** (o pulsando con el botón derecho del ratón sobre su nombre y seleccionando la opción **Cambiar nombre de Bus::remove\_students**). Esto mostrará un diálogo donde podrá introducir el nuevo nombre de la función y donde también podrá ver todos los lugares donde se está usando realmente la función:

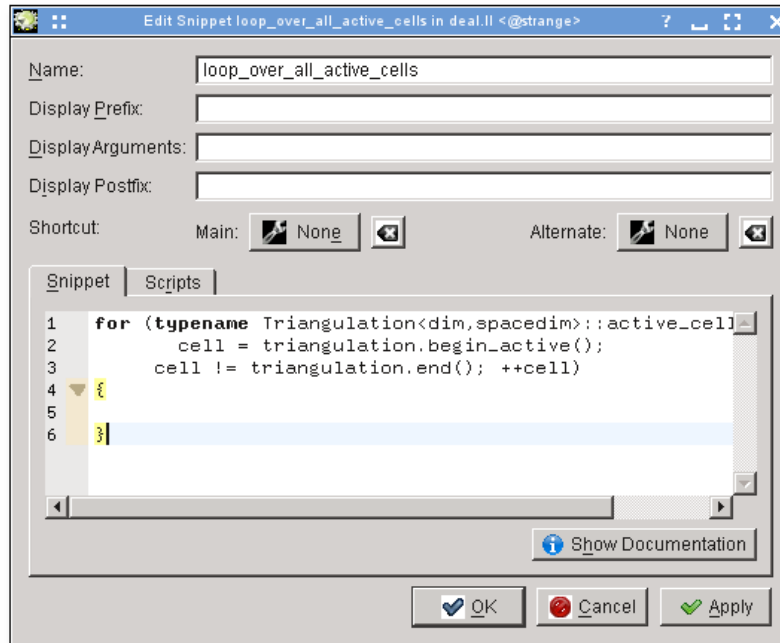


### 3.4.5. Fragmentos de código

La mayoría de los proyectos contienen trozos de código que es necesario escribir frecuentemente en el código fuente. Algunos ejemplos son: para los escritores de compiladores, un bucle para todas las instrucciones; para los escritores de interfaces de usuario, comprobaciones de que la entrada del usuario es válida y, en caso contrario, mostrar un mensaje de error; en el proyecto del autor de estas líneas, sería código del tipo

```
for (typename Triangulation::active_cell_iterator
     cell = triangulation.begin_active();
     cell != triangulation.end(); ++cell)
    &#8230; hacer algo con la celda&#8230;
```

En lugar de teclear este tipo de texto una y otra vez (con los errores tipográficos repetitivos que se suelen cometer), la herramienta de **Fragmentos de código** de KDevelop puede serle de ayuda. Para ello, abra el visor de la herramienta (consulte [Herramientas y visores](#) si el botón correspondiente aún no está presente en el borde de la ventana). A continuación, pulse el botón 'Añadir repositorio' (un nombre poco apropiado, ya que le permite crear una colección identificada por su nombre que contiene fragmentos de código fuente para un propósito particular, como, por ejemplo, código fuente de C++) para crear un repositorio vacío. Pulse después **+** para añadir un fragmento de código, con lo que se le mostrará un diálogo como el siguiente:

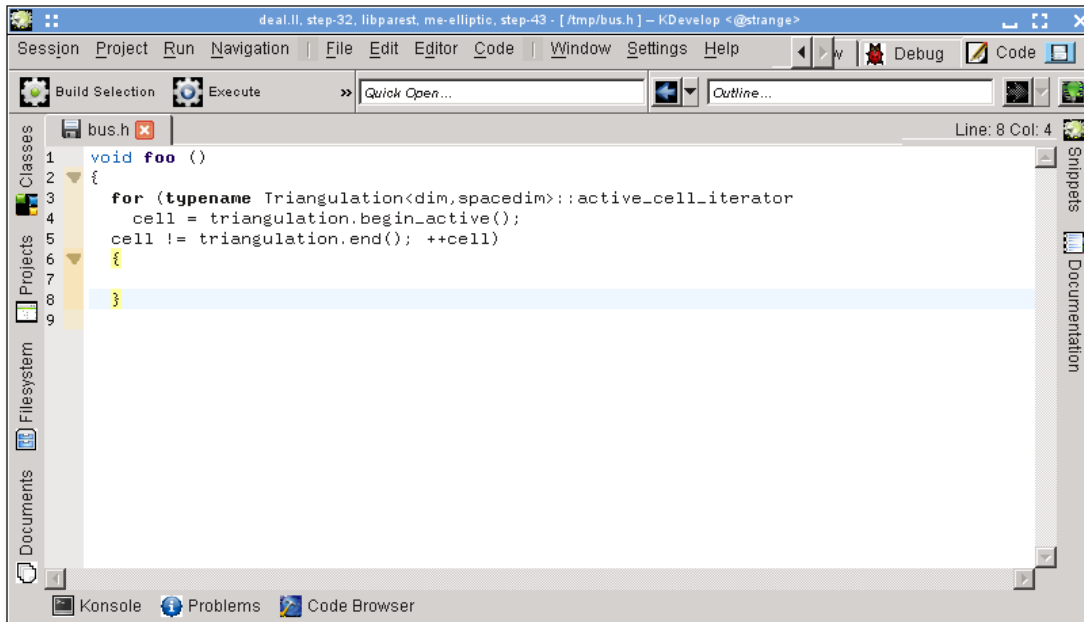


**NOTA**

El nombre del fragmento no debe contener espacios ni otros caracteres especiales, pues debe parecerse a cualquier nombre de función o variable normal (por motivos que se aclararán en el siguiente párrafo)

Para usar un fragmento definido de este modo cuando esté editando código fuente, escriba el nombre de dicho fragmento como haría con cualquier otro nombre de función o de variable. Este nombre también estará disponible para la terminación automática de texto (lo que significa que no hay ningún peligro al usar un nombre largo y descriptivo para un fragmento, como el anterior), por lo que cuando acepte la sugerencia de la ayuda emergente de terminación automática (por ejemplo, pulsando la tecla **Intro**), la parte del nombre del fragmento ya introducida se sustituirá con todo el texto que contiene el fragmento de código, que se mostrará probablemente sangrado:

## Manual de KDevelop

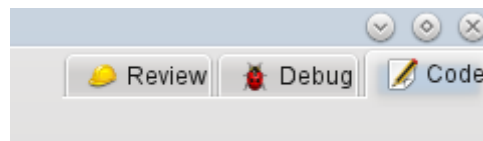


Tenga en cuenta que para que esto funcione el visor de la herramienta **Fragmentos de código** no debe estar abierto ni visible: solamente necesitará el visor de esta herramienta para definir nuevos fragmentos de código. Un modo alternativo, aunque menos conveniente, de expandir un fragmento de código consiste en pulsar sobre él en el visor de la herramienta.

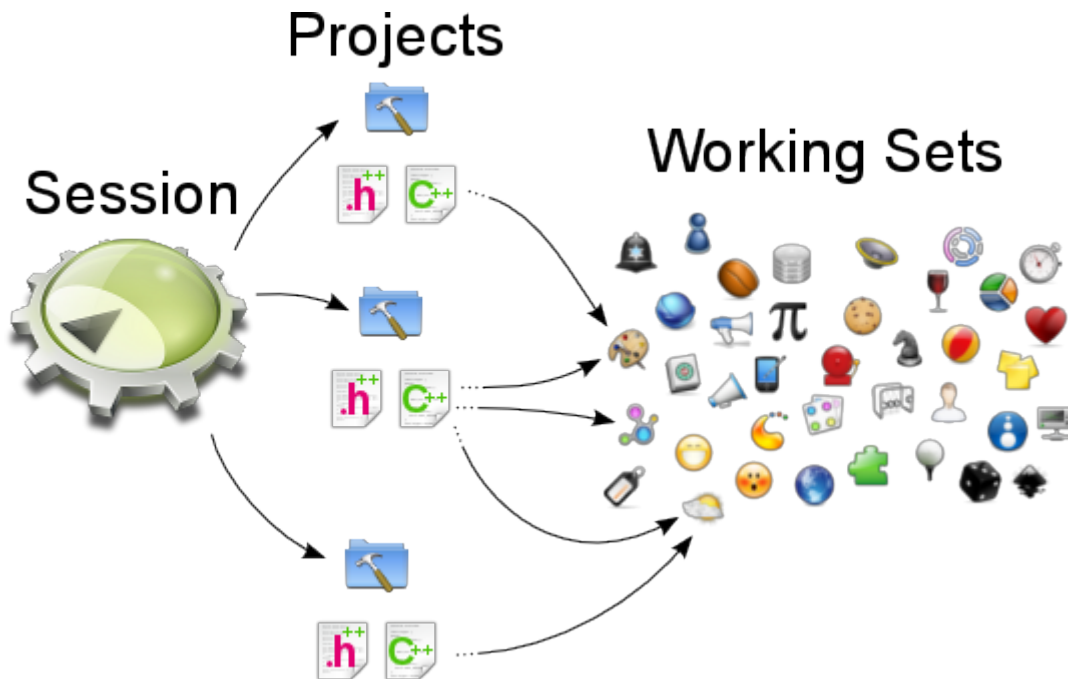
### NOTA

Los fragmentos de código son mucho más potentes de lo que se ha explicado. Para una descripción completa de lo que se puede hacer con ellos, consulte la [documentación detallada de la herramienta de fragmentos de código](#).

## 3.5. Modos y conjuntos de trabajo

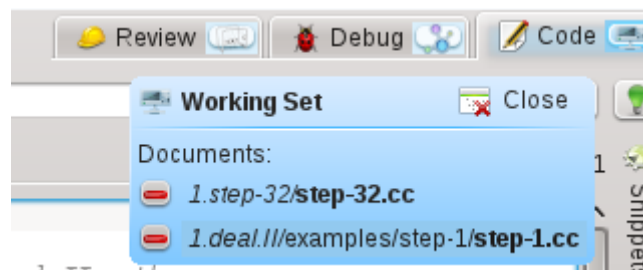


Si ha llegado hasta aquí, eche un vistazo a la parte superior derecha de la ventana principal de KDevelop. Como se muestra en la imagen, verá que existen tres **modos** en los que puede encontrarse KDevelop: **Código** (el modo que describimos en este capítulo sobre el trabajo con código fuente), **Depurar** (consulte [Depuración de programas](#)) y **Revisar** (consulte [Trabajar con sistemas de control de versiones](#)).



Cada modo tiene su propio conjunto de herramientas apiladas en el perímetro de la ventana principal, así como un *conjunto de trabajo* de archivos y documentos actualmente abiertos. Aún más, cada uno de estos conjuntos de trabajo está asociado con una sesión actual; es decir, tenemos la relación mostrada anteriormente. Tenga en cuenta que los archivos del conjunto de trabajo proceden de la misma sesión, aunque pueden pertenecer a diferentes proyectos que sean parte de la misma sesión.

Cuando abre KDevelop por primera vez, el conjunto de trabajo está vacío (no hay ningún archivo abierto). Pero, a medida que va abriendo archivos para editarlos (o para depurarlos o revisarlos en los otros modos), el conjunto de trabajo va creciendo. Cuando su conjunto de trabajo no esté vacío verá un símbolo en la pestaña, como se muestra más abajo. También notará que cuando cierre KDevelop y vuelva a abrirlo otra vez, el conjunto de trabajo se habrá guardado y restaurado; es decir, tendrá el mismo conjunto de archivos abiertos.



Si sitúa el puntero del ratón sobre el símbolo del conjunto de trabajo, obtendrá una ventana emergente que le mostrará los archivos actualmente abiertos en dicho conjunto de trabajo (en este caso, los archivos `step-32.cc` y `step-1.cc`). Si pulsa en el signo menos de color rojo se cerrará la pestaña del archivo correspondiente. Tal vez más importante, si pulsa sobre el correspondiente botón que lo identifica, podrá **cerrar** todo el conjunto de trabajo a la vez (es decir, cerrará todos los archivos actualmente abiertos). La importancia de cerrar un conjunto de trabajo, no obstante, reside en que no solo se cierran todos los archivos que lo componen, sino que realmente se guardan todos los archivos y se abre un nuevo conjunto de trabajo vacío. Puede ver esto a continuación:



Note los dos símbolos a la izquierda de las pestañas de los tres modos (el corazón y el símbolo desconocido que hay a su izquierda). Cada uno de estos símbolos representa un conjunto de trabajo guardado, además del conjunto de trabajo actualmente abierto. Si sitúa el puntero del ratón sobre el símbolo del corazón, obtendrá algo semejante a lo siguiente:



Le muestra que el conjunto de trabajo correspondiente contiene dos archivos y sus correspondientes nombres de proyecto: Makefile y changes.h. Si pulsa **Cargar** se cerrará y guardará el conjunto de trabajo actual (que, como se muestra aquí, contiene los archivos abiertos tria.h y tria.cc) y se abrirá en su lugar el conjunto de trabajo seleccionado. También puede borrar de forma permanente un conjunto de trabajo, lo que hace que desaparezca de la lista de conjuntos de trabajo guardados.

### 3.6. Algunos atajos de teclado útiles

El editor de KDevelop sigue los atajos de teclado típicos para todas las operaciones normales de edición. No obstante, también permite usar cierto número de operaciones más avanzadas cuando se edita código fuente, muchas de las cuales están ligadas a ciertas combinaciones particulares de teclas. Las siguientes son particularmente interesantes en muchos casos:

Saltar a través del código	
<b>Ctrl-Alt-O</b>	Apertura rápida de archivo: introduzca parte de un nombre de archivo y seleccione en todos árboles de directorios de los archivos de los proyectos de la sesión actual que coincidan con la cadena introducida: se abrirá el archivo que seleccione.
<b>Ctrl-Alt-C</b>	Apertura rápida de clase: introduzca parte del nombre de una clase y seleccione entre todos los nombres de clases que coincidan; el cursor saltará a la declaración de la clase que seleccione.
<b>Ctrl-Alt-M</b>	Apertura rápida de función: introduzca parte del nombre de una función (miembro) y seleccione entre los nombres que coincidan; tenga en cuenta que la lista muestra tanto declaraciones como definiciones, y que el cursor saltará al elemento seleccionado.

## Manual de KDevelop

<b>Ctrl-Alt-Q</b>	Apertura rápida universal: escriba cualquier cosa (nombre de archivo, nombre de clase, nombre de función) para obtener una lista de todo lo que coincida para poder seleccionarlo.
<b>Ctrl-Alt-N</b>	Esquema: Proporciona una lista de todas las cosas que están ocurriendo en este archivo. Por ejemplo, declaraciones de clases y definiciones de funciones.
<b>Ctrl-,</b>	Saltar a la definición de una función si el cursor está actualmente sobre la declaración de una función
<b>Ctrl-.</b>	Saltar a la declaración de una función o variable si el cursor está actualmente en una definición de función
<b>Ctrl-Alt-RePág</b>	Saltar a la siguiente función
<b>Ctrl-Alt-AvPág</b>	Saltar a la función anterior
<b>Ctrl-G</b>	Ir a una línea

<b>Búsqueda y sustitución</b>	
<b>Ctrl-F</b>	Buscar
<b>F3</b>	Encontrar siguiente
<b>Ctrl-R</b>	Sustituir
<b>Ctrl-Alt-F</b>	Buscar o sustituir en múltiples archivos

<b>Otras cosas</b>	
<b>Ctrl-_</b>	Plegar un nivel: compacta un bloque haciendo que no se visualice, por ejemplo, cuando desea concentrarse en una visión más amplia dentro de una función
<b>Ctrl-+</b>	Expande un nivel: deshace el plegado anterior
<b>Ctrl-D</b>	Comentar el texto seleccionado o la línea actual
<b>Ctrl-Mayúsculas-D</b>	Comentar el texto seleccionado o la línea actual
<b>Alt-Mayúsculas-D</b>	Documentar la función actual. Si el cursor está sobre una declaración de función o de clases, al pulsar esta tecla se creará un comentario predefinido al estilo de «doxygen» que contiene una lista con todos los parámetros, valores de retorno, etc.
<b>Ctrl-T</b>	Intercambia los caracteres actual y anterior
<b>Ctrl-K</b>	Borrar la línea actual (nota: no es como en emacs 'borrar desde aquí hasta el final de la línea')

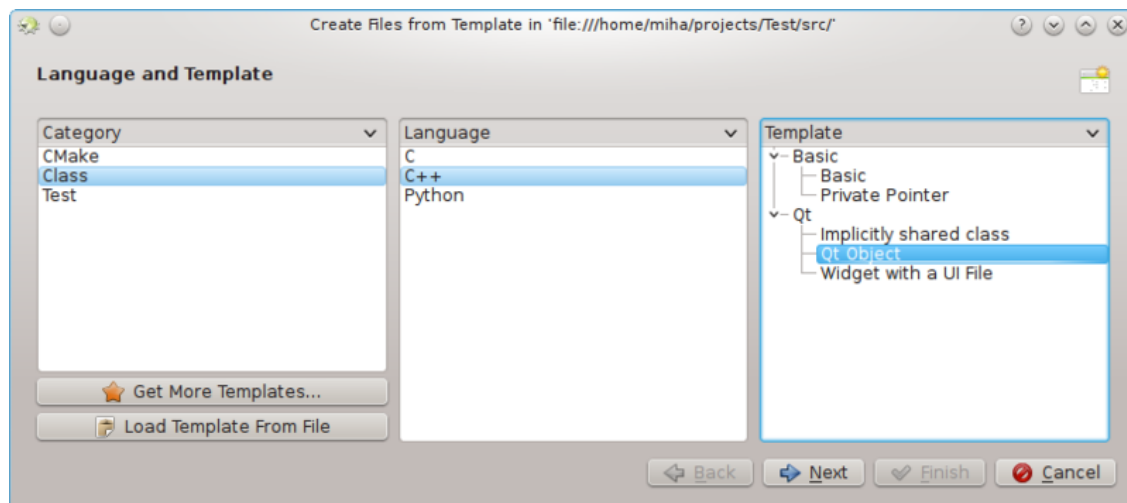
## Capítulo 4

# Generación de código con plantillas

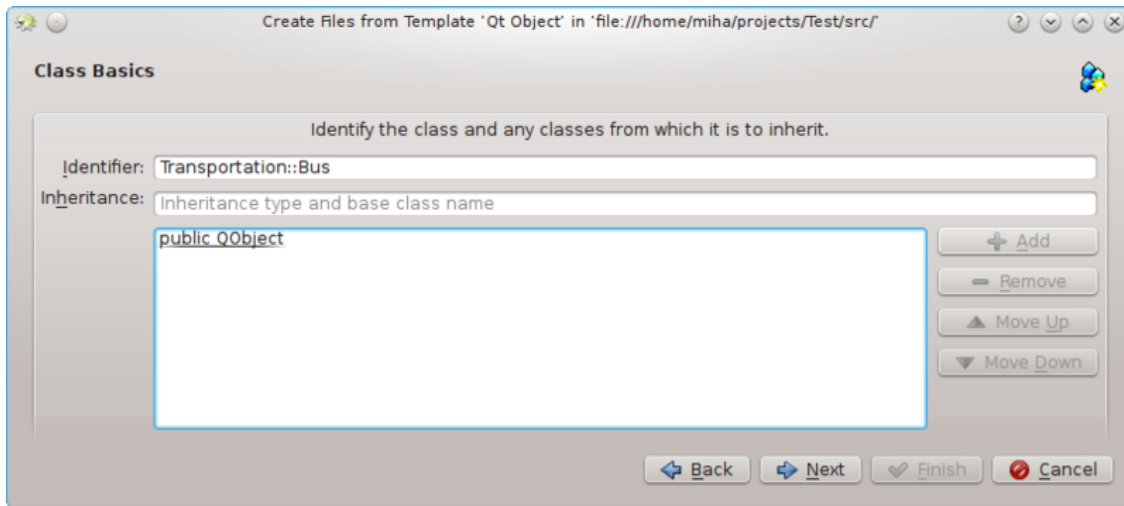
KDevelop usa plantillas para generar archivos de código fuente y para evitar tener que escribir código repetitivo.

### 4.1. Creación de una nueva clase

El uso más común para la generación de código es probablemente la escritura de nuevas clases. Para crear una nueva clase en un proyecto existente, pulse con el botón derecho del ratón en una carpeta del proyecto y elija **Crear desde plantilla...**. El mismo diálogo se puede iniciar desde el menú pulsando **Archivo** → **Nuevo desde plantilla...**, aunque si usa una carpeta del proyecto se ahorrará tener que fijar una URL base para los archivos de salida. Elija **Clase** en el visor de selección de categorías y luego el idioma y la plantilla en las otras dos vistas. Tras seleccionar una plantilla para la clase tendrá que especificar los detalles de la nueva clase.

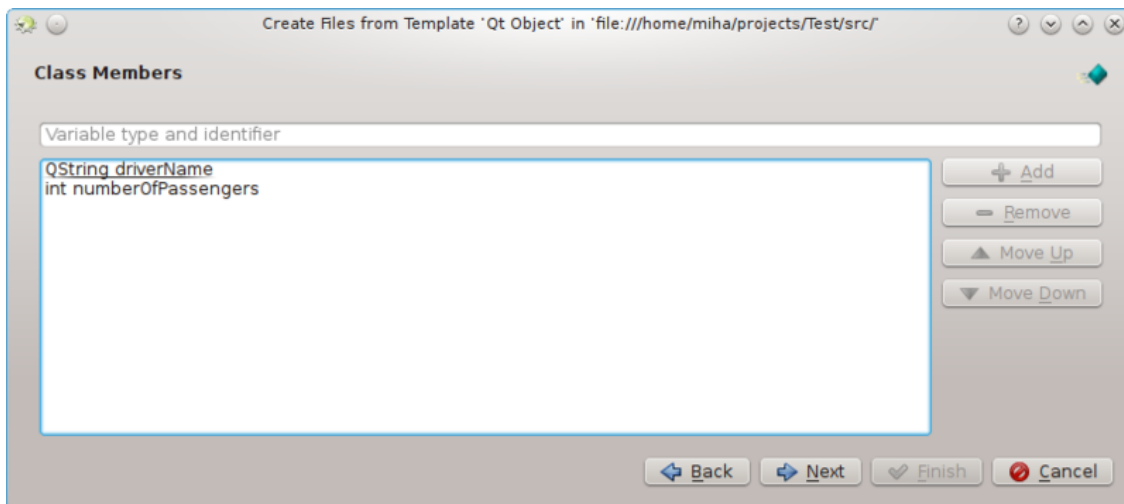


En primer lugar, tendrá que especificar un identificador para la nueva clase. Puede ser un nombre sencillo (como `Bus`) o un identificador completo con espacios de nombres (como `Transporte::Bus`). En el último caso, KDevelop analizará el identificador y separará correctamente los espacios de nombres del nombre real de la clase. En la misma página podrá añadir las clases base para la nueva clase. Es posible que note que algunas plantillas escogen una clase base por sí mismas; si no son de su agrado, puede eliminarlas y/o añadir otras bases. Aquí debe escribir la sentencia de herencia completa, que es dependiente del lenguaje, como `public QObject` para C++, `extends algunaClase` para PHP, o simplemente el nombre de la clase para Python.



En la siguiente página se le ofrecerá una selección de los métodos virtuales de todas las clases heredadas, así como algunos constructores, destructores y operadores por omisión. Si marca la casilla que hay junto a cada firma de un método se implementará dicho método en la nueva clase.

Si pulsa **Siguiente** se mostrará una página en la que puede añadir miembros a la clase. Dependiendo de la plantilla seleccionada, pueden aparecer en la nueva clase como variables miembro, o la plantilla puede crear propiedades con sus respectivos «setters» y «getters». En un lenguaje para el que los tipos de variables tienen que estar declarados, como C++, debe especificar tanto el tipo como el nombre del miembro, como `int numero` o `QString nombre`. Para otros lenguajes puede olvidarse del tipo, aunque es una buena práctica introducirlo siempre, ya que la plantilla seleccionada puede seguir haciendo uso de él.



En las siguientes páginas podrá elegir una licencia para la nueva clase, fijar cualquier opción personalizada que necesite la plantilla seleccionada y configurar las ubicaciones de salida para todos los archivos generados. Si pulsa **Finalizar** completará el asistente y se creará la nueva clase. Los archivos generados se abrirán en el editor, por lo que puede comenzar a añadir código fuente en ellos enseguida.

Tras crear una nueva clase de C++ se le ofrecerá la opción de añadir la clase a un proyecto de destino. Elija un destino en la página del diálogo, o descarte la página y añada los archivos a un destino de forma manual.

Si elige la plantilla Objeto de Qt, marca algunos de los métodos por omisión y añada dos variables miembro, la salida debería parecerse a la de la siguiente imagen.

```

Bus.h
Bus.cpp

/*
 * This file is licensed under the Free Transportation License 3.14
 */

#ifndef TRANSPORTATION_BUS_H
#define TRANSPORTATION_BUS_H

#include <QtCore/QObject>

namespace Transportation {

class BusPrivate;

class Bus : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString driverName READ driverName WRITE setDriverName)
    Q_PROPERTY(int numberOfPassengers READ numberOfPassengers WRITE setNumberOfPassengers)

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

    QString driverName() const;
    int numberOfPassengers() const;

public Q_SLOTS:
    void setDriverName(const QString& driverName);
    void setNumberOfPassengers(int numberOfPassengers);

private:
    Q_DECLARE_PRIVATE(Bus)
};
}

#endif // TRANSPORTATION_BUS_H

```

Puede ver que los datos miembros se convierten en propiedades de Qt, con funciones accesoras y las macros `Q_PROPERTY`. Los argumentos para las funciones «setters» se pasan como referencias a constantes cuando sea apropiado. Además, se declara una clase privada, así como un puntero privado creado con `Q_DECLARE_PRIVATE`. Todo esto lo hace la plantilla. Si elige una plantilla diferente en el primer paso, la salida puede ser muy distinta.

## 4.2. Creación de una nueva prueba unitaria

A pesar de que la mayoría de las infraestructuras de pruebas necesitan que cada prueba también sea una clase, KDevelop incluye un método para simplificar la creación de pruebas unitarias. Para crear una nueva prueba, haga clic con el botón derecho del ratón en la carpeta de un proyecto y elija **Crear desde plantilla...** En la página de selección de plantillas, escoja *Prueba* como categoría, y a continuación seleccione el lenguaje de programación y la plantilla, y luego pulse **Siguiente**.

Se le preguntará el nombre de la prueba y por una lista de casos de pruebas. Para los casos de pruebas, solo tiene que especificar una lista de nombre. Algunas infraestructuras de pruebas unitarias, como PyUnit y PHPUnit, necesitan que los casos de pruebas comiencen por un prefijo especial. En KDevelop, la plantilla es la responsable de añadir este prefijo, por lo que no tiene que añadir dicho prefijo a los casos de pruebas en este paso. Tras pulsar **Siguiente**, indique la licencia y las ubicaciones de salida para los archivos generados, con lo que la prueba terminará de crearse.

Las pruebas unitarias creadas de este modo no se añadirán a ningún objetivo de forma automática. Si está usando CTest u otra infraestructura de pruebas, asegúrese de añadir los archivos nuevos al objetivo.

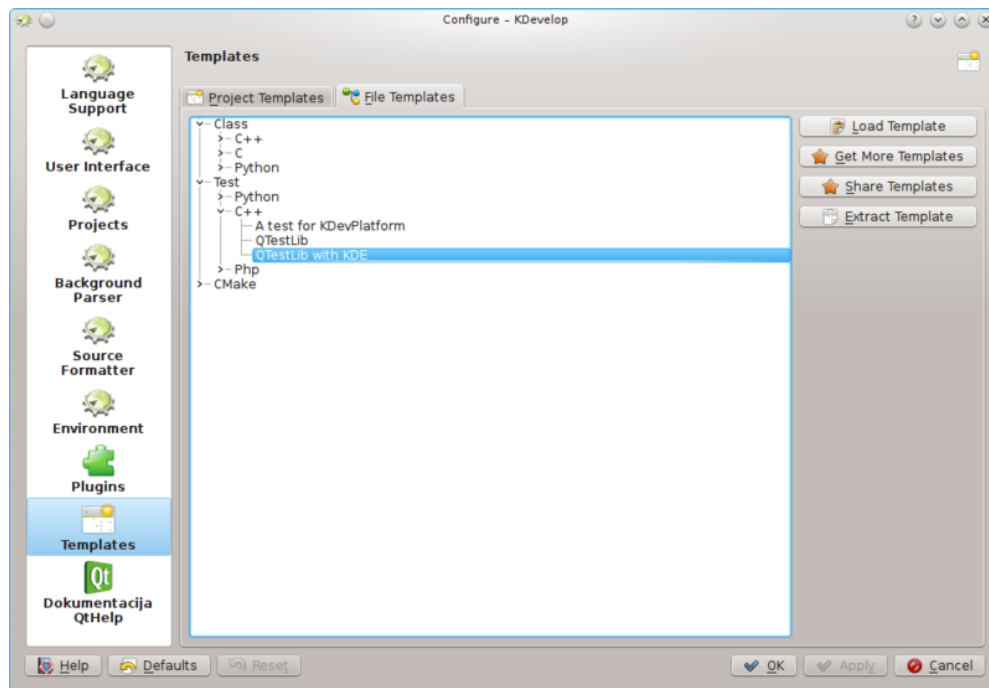
## 4.3. Otros archivos

A pesar de que las clases y las pruebas unitarias reciben una atención especial cuando generan código a partir de plantillas, se puede usar el mismo método para cualquier tipo de archivos de código fuente. Por ejemplo, es posible usar una plantilla para el módulo Find de CMake o

para un archivo «.desktop». Para ello, elija **Crear desde plantilla...** y seleccione la categoría y la plantilla deseadas. Si la categoría seleccionada no es `Clase` ni `Prueba`, solo dispondrá de la opción de escoger la licencia, cualquier opción personalizada indicada por la plantilla y la ubicación de los archivos de salida. Como ocurre con las clases y con las pruebas, al terminar el asistente se generarán los archivos y se abrirán en el editor.

## 4.4. Gestión de plantillas

Desde el asistente que se ejecuta al seleccionar **Archivo** → **Nuevo desde plantilla...** también puede descargar plantillas de archivo adicionales pulsando el botón **Obtener nuevas plantillas...**. Esto abre el diálogo «Obtener novedades», desde donde podrá instalar plantillas adicionales, así como actualizar o eliminar las existentes. También dispone de un módulo de configuración para las plantillas, al que puede llegar seleccionando **Preferencias** → **Configurar KDevelop** → **Plantillas**. Desde aquí puede gestionar tanto las plantillas de archivo (anteriormente explicadas), como las plantillas de proyecto (usadas para crear nuevos proyectos).



Por supuesto, si ninguna de las plantillas disponibles se ajusta a su proyecto, siempre puede crear otras nuevas. El modo más sencillo de hacerlo consiste, probablemente, en copiar y modificar una plantilla existente, aunque también dispone de un breve [tutorial](#) y de un [documento con la especificación](#) más extenso a modo de ayuda. Para copiar una plantilla instalada, abra el gestor de plantillas usando **Preferencias** → **Configurar KDevelop;...** → **Plantillas**, seleccione la plantilla que desea copiar y pulse luego el botón **Extraer plantilla**. Seleccione una carpeta de destino y pulse **Aceptar** para extraer el contenido de la plantilla en la carpeta indicada. Ahora podrá editar la plantilla abriendo los archivos extraídos y modificándolos. Cuando haya terminado, puede importar la nueva plantilla en KDevelop abriendo el gestor de plantillas, yendo a la pestaña apropiada (que puede ser **Plantillas de proyecto** o **Plantillas de archivos**) y pulsando **Cargar plantilla**. Abra el archivo de descripción de la plantilla, que es el que tiene una de las extensiones `.kdevtemplate` o `.desktop`. KDevelop comprimirá estos archivos en un archivo comprimido de plantilla y luego importará la plantilla.

**NOTA**

Cuando copie una plantilla existente, asegúrese de que cambia su nombre antes de importarla de nuevo. En caso contrario sobrescribirá la plantilla existente o acabará con dos plantillas con el mismo nombre. Para cambiar el nombre de una plantilla, cambie el nombre del archivo de descripción para que sea único (pero conserve el sufijo) y modifique la entrada `Name` del archivo de descripción.

Si desea crear una plantilla desde cero, puede comenzar con una plantilla de clase de C++ de ejemplo mediante la [creación de un nuevo proyecto](#), seleccionando el proyecto `Plantilla` de clase de C++ en la categoría `KDevelop`.

## Capítulo 5

# Construir (compilar) proyectos con Makefiles personalizados

Muchos proyectos describen cómo se deben compilar los archivos de código fuente (y qué archivos se deben recompilar tras realizar cambios en un archivo de código fuente o de cabecera) usando Makefiles que son interpretados por el programa **make** (vea, por ejemplo, [GNU make](#)). Para proyectos sencillos, a menudo resulta más fácil configurar dicho archivo a mano. Los proyectos más grandes suelen integrar sus Makefiles con las **herramientas automáticas de GNU** (autoconf, autoheader y automake). En esta sección, supondremos que dispone de un archivo Makefile para su proyecto y que desea indicarle a KDevelop cómo debe proceder con él.

### NOTA

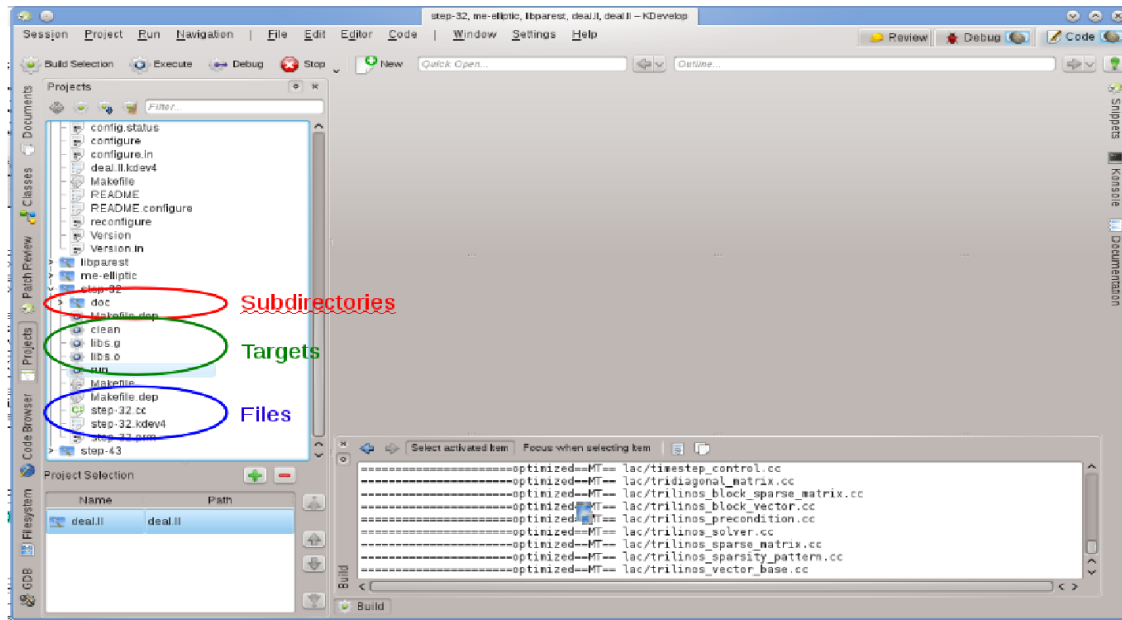
KDevelop 4.x no sabe nada sobre las **herramientas automáticas de GNU** en el momento en que se ha escrito esta sección. Si su proyecto las utiliza, tendrá que ejecutar `./configure` u otra orden relacionada de forma manual en la consola. Si prefiere hacer esto desde KDevelop, abra la herramienta **Konsole** (si es necesario, añádala al borde de la ventana principal usando el menú **Ventanas** → **Añadir vista de herramienta**), que le proporciona acceso a una vista de la ventana de la consola, y ejecute `./configure` desde la línea de órdenes de dicha vista.

El primer paso consiste en indicarle a KDevelop cuáles son los objetivos de sus Makefiles. Existen dos modos de hacer esto: seleccionar objetivos de Makefiles individuales, o escoger un conjunto de objetivos que desee construir con frecuencia. Para ambas aproximaciones, abra la herramienta **Proyectos** pulsando en el botón **Proyectos** que hay en el borde de la ventana principal (si no tiene este botón, añádalo como se indicó con anterioridad). La ventana de la herramienta **Proyectos** consta de dos partes: la mitad superior, (con el título **Proyectos**) donde se listan todos los proyectos y se le permite desplegar los árboles de directorios que contiene; y la mitad inferior (con el título **Selección de proyecto**) que lista un subconjunto de los proyectos que se construirán cuando use el elemento del menú **Proyecto** → **Construir selección** o pulse **F8**. Más adelante volveremos a esta cuestión.

### 5.1. Construcción de objetivos Makefile individuales

En la parte superior del visor del proyecto, despliegue el árbol de un proyecto (por ejemplo, del que desea ejecutar un objetivo Makefile particular). Se mostrarán iconos para: (i) los directorios que hay bajo el proyecto, (ii) los archivos que hay en el directorio superior de dicho proyecto, y (iii) los objetivos Makefile que KDevelop puede identificar. Estas categorías se muestran en

la imagen de la derecha. Tenga presente que KDevelop *entiende* la sintaxis de Makefile hasta cierto punto, por lo que puede identificar objetivos definidos en dicho Makefile (aunque este conocimiento tiene sus límites si los objetivos están compuestos o si son implícitos).





Para construir cualquiera de los objetivos que se listan ahí, pulse sobre él con el botón derecho del ratón y seleccione **Construir**. Por ejemplo, si hace esto con el objetivo 'clean' se ejecutará 'make clean'. Puede ver el desarrollo de la ejecución en la ventana que se abre con el título **Construir**, que muestra la orden y su salida. (Esta ventana corresponde a la herramienta **Construir**, y puede cerrarla y abrirla posteriormente usando el botón de la herramienta **Construir** que hay en el borde de la ventana principal. Esto se muestra en la parte inferior derecha de la imagen).

## 5.2. Selección de una colección de Makefiles de destino para compilación repetitiva

Si pulsa con el botón derecho del ratón sobre objetivos Makefile individuales cada vez que quiera construir algo quedará obsoleto con rapidez. Más bien, sería deseable tener objetivos individuales para uno o más de los proyectos de la sesión que necesitamos compilar de forma repetitiva sin demasiado uso del ratón. Aquí es donde interviene el concepto de 'Construir selecciones de objetivos': consiste en una colección de objetivos Makefile que se construyen uno tras otro cada vez que pulse el botón **Construir selección** que hay en la lista de botones superiores, o cuando use la opción del menú **Proyecto** → **Construir selección** o pulse la tecla de función **F8**.

La lista de los Makefiles de destino seleccionada se muestra en la mitad inferior del visor de la herramienta **Proyectos**.

Por omisión, la selección contiene todos los proyectos, aunque puede cambiarla. Por ejemplo, si la lista de proyectos contiene tres proyectos (una biblioteca base, L, y dos aplicaciones, A y B), pero solo en la actualidad solamente trabaja en el proyecto A, tal vez desee eliminar el proyecto B de la selección resaltándolo y pulsando el botón . Además, también es probable que quiera asegurarse de que la biblioteca L se construye antes que el proyecto A subiendo o bajando las entradas de la selección con los botones que hay a la derecha de la lista. También puede obtener un objetivo Makefile particular en la lista de selección haciendo clic sobre él con el botón derecho

del ratón y seleccionando la opción **Añadir al conjunto de construcción**, o bien resaltándolo y pulsando el botón  que hay justo encima de la lista de objetivos seleccionados.

KDevelop le permite configurar lo que se va a hacer cada vez que construya la selección. Para este fin, use la opción del menú **Proyecto** → **Abrir configuración**. Ahí podrá, por ejemplo, seleccionar el número de trabajos simultáneos que debe ejecutar 'make' (si su equipo dispone, por ejemplo, de 8 núcleos de procesador, una buena elección sería introducir 8 en este campo). En este diálogo, el **Objetivo make por omisión** es un objetivo Makefile que se usa para *todos* los objetivos de la selección.

### 5.3. Qué hacer con los mensajes de error

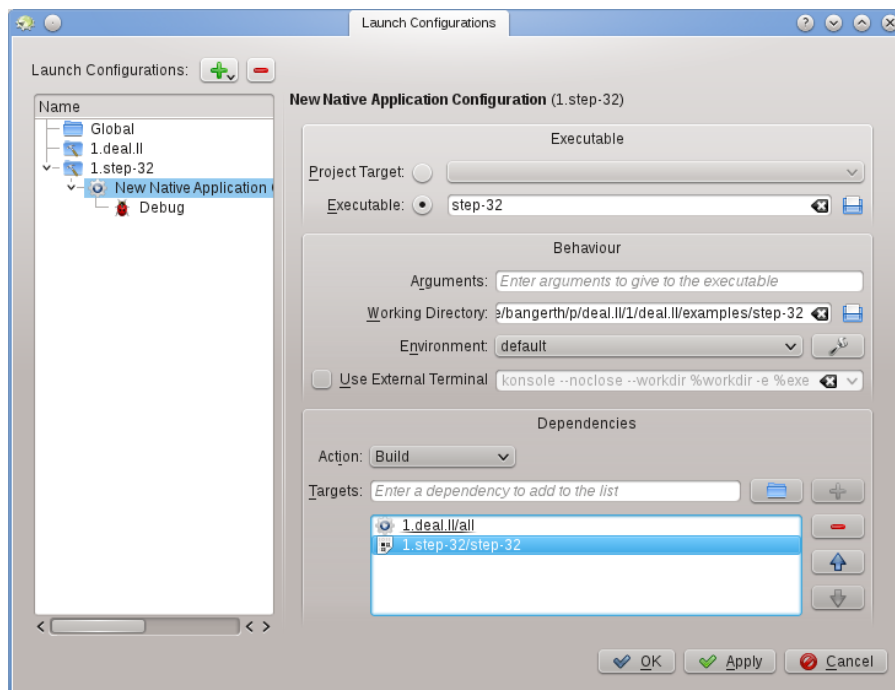
Si el compilador encuentra un mensaje de error, haga clic en la línea del mensaje de error para que el editor salte a la línea (y columna, si está disponible) donde se ha generado el error. Dependiendo del mensaje de error, KDevelop también le ofrecerá diversas acciones posibles para solucionarlo (por ejemplo, declarar una variable previamente no declarada si se encuentra un símbolo desconocido).

## Capítulo 6


# Ejecutar programas en KDevelop

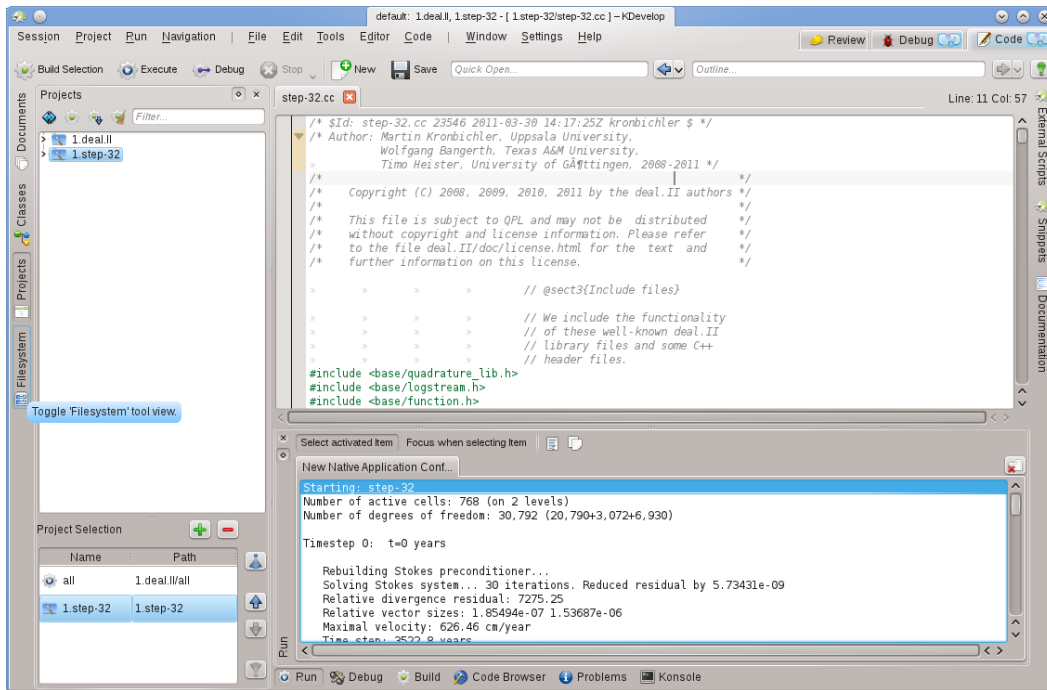
Tras construir un programa, es posible que desee ejecutarlo. Para ello, necesitará configurar *lanzadores* para sus proyectos. Un *lanzador* consiste en el nombre de un ejecutable, un conjunto de parámetros de la línea de órdenes y un entorno de ejecución (como ‘ejecutar este programa en la consola’ o ‘ejecutar este programa en el depurador’).

### 6.1. Configuración de lanzadores en KDevelop



Para configurarlos, vaya a la opción del menú **Ejecutar** → **Configurar lanzadores**, seleccione el proyecto para el que quiera añadir un lanzador y haga clic en el botón **+**. Introduzca a continuación el nombre del ejecutable y la ruta donde desee ejecutar el programa. Si la ejecución del programa depende de que se primero se construyan el ejecutable y otras bibliotecas, es probable que quiera añadirlos a la lista que hay en la parte inferior: seleccione **Construir** en el menú

desplegable, luego pulse el símbolo  que hay a la derecha del campo de texto y seleccione los objetivos que desee construir. En el ejemplo anterior, hemos seleccionado el objetivo **todo** del proyecto *1.deal.II* y *step-32* del proyecto *1.setp-32* para asegurar que tanto la biblioteca base como el programa de la aplicación se han compilado y que están actualizados antes de ejecutar el programa. Cuando esté aquí, es posible que también quiera configurar un lanzador para depuración pulsando sobre el símbolo **Depurar** y añadiendo el nombre del programa de depuración; si es el depurador por omisión del sistema (por ejemplo, gdb en Linux<sup>®</sup>), no necesita realizar este paso.



Ahora puede probar a ejecutar el programa: seleccione **Ejecutar** → **Ejecutar lanzador** en el menú de la ventana principal de KDevelop (o pulse **Mayúsculas-F9**) para que el programa se ejecute en una ventana independiente de KDevelop. La imagen superior muestra el resultado: la nueva ventana de la herramienta **Ejecutar** en la parte inferior muestra la salida del programa que se está ejecutando (en este caso, *step-32*).

**NOTA**  
 Si ha configurado diversos lanzadores, puede elegir cuál de ellos se debe ejecutar cuando pulse **Mayúsculas-F9** yendo a **Ejecutar** → **Configuración del lanzador actual**. No obstante, existe una forma no obvia de editar el nombre de una configuración: en el diálogo que se muestra cuando selecciona **Ejecutar** → **Configuración del lanzador actual**, haga doble clic sobre el nombre de la configuración en la vista de árbol que hay a la izquierda, lo que le permitirá cambiar el nombre de la configuración.

## 6.2. Algunos atajos de teclado útiles

Ejecución de un programa	
<b>F8</b>	Construir (llamar a make)
<b>Mayúsculas-F9</b>	Ejecutar

<b>Alt-F9</b>	Ejecutar el programa en el depurador; es posible que desee fijar puntos de interrupción con anterioridad, por ejemplo, pulsando con el botón derecho del ratón sobre una determinada línea en el código fuente
---------------	--

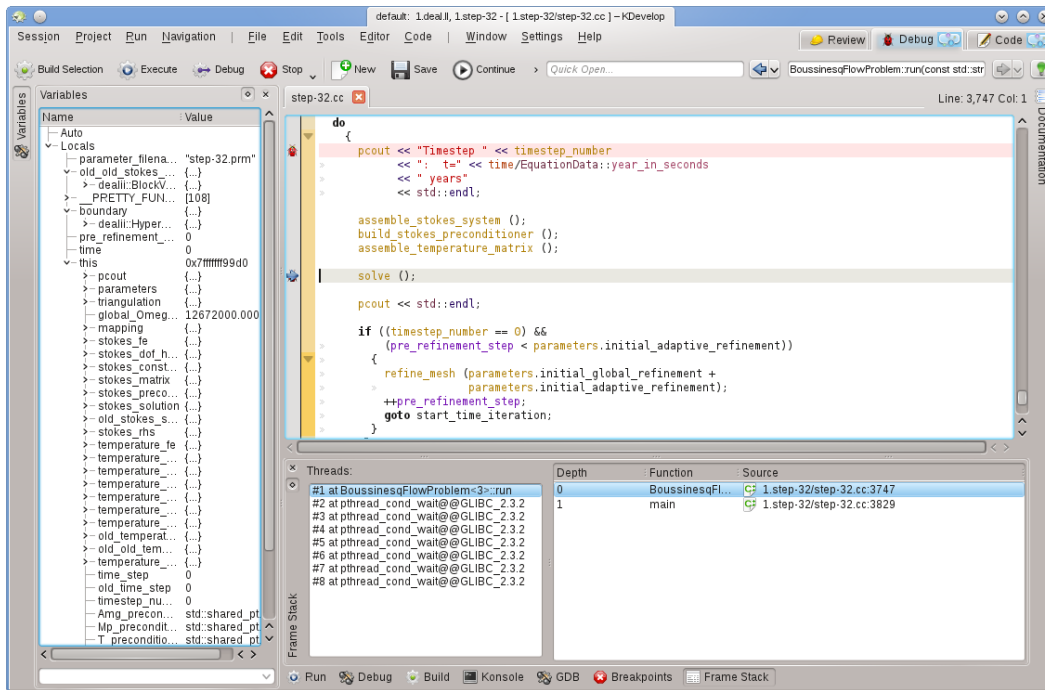
## Capítulo 7

# Depuración de programas en KDevelop

### 7.1. Ejecutar un programa en el depurador

Tras configurar un lanzador (consulte [Ejecución de programas](#)), también podrá ejecutarlo en un depurador: seleccione la opción del menú **Ejecutar** → **Lanzador de depuración** o pulse **Alt-F9**. Si está familiarizado con `gdb`, el resultado es el mismo que se obtiene al iniciar `gdb` con el nombre del ejecutable indicado en la configuración del lanzador y usando luego **Ejecutar**. Esto significa que si el programa llama en algún momento a `abort()` (por ejemplo, cuando se genera una aserción no contemplada) o si se produce un error de segmentación, se detendrá el depurador. Por otra parte, si el programa se ejecuta hasta el final (de forma correcta o incorrecta), el depurador no se detendrá por sí mismo antes de que el programa haya finalizado. En este último caso, antes de ejecutar el lanzador de depuración, querrá fijar un punto de interrupción en todas las líneas del código fuente donde desee que se detenga el depurador. Puede hacer esto moviendo el cursor a cada una de dichas líneas y seleccionando la opción del menú **Ejecutar** → **Conmutar punto de interrupción**, o haciendo clic con el botón derecho del ratón en una línea y seleccionando **Conmutar punto de interrupción** en el menú de contexto.

## Manual de KDevelop

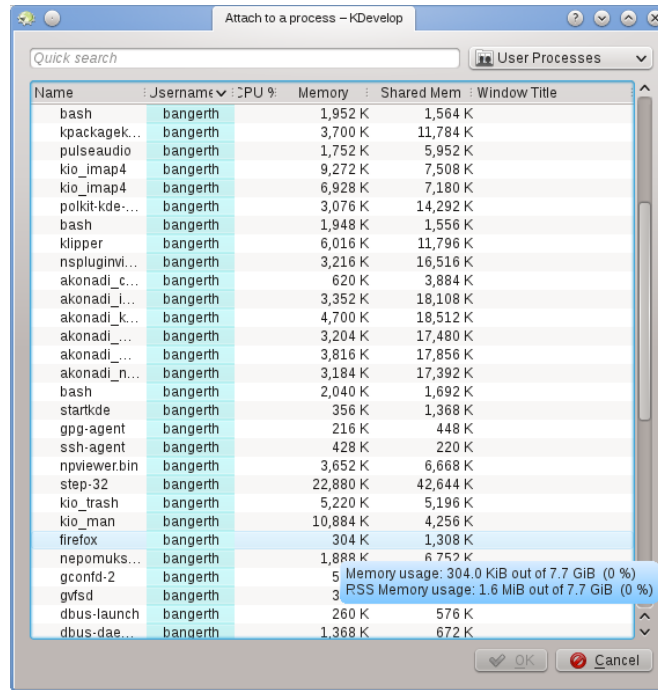


La ejecución de un programa en el depurador hará que KDevelop entre en un modo diferente: sustituirá todos los botones de 'Herramientas' del borde de la ventana principal por otros más adecuados para la depuración que para la edición. Puede ver en qué modo se encuentra observando la parte superior derecha de la ventana, donde existen pestañas denominadas **Revisar**, **Depurar** y **Código**. Si pulsa sobre ellas podrá cambiar entre los tres modos. Cada modo tiene su propio juego de vistas de herramientas, que puede configurar del mismo modo que hicimos con las herramientas de **Código** en la sección [Herramientas y vistas](#).

Una vez que el depurador se detiene (en un punto de interrupción o donde se haya llamado a `abort()`), puede inspeccionar diversa información sobre el programa. Por ejemplo, en la imagen superior, hemos seleccionado la herramienta **Pila de ejecución** en la parte inferior (aproximadamente, como las órdenes 'backtrace' e 'info threads' de gdb), que muestra en la parte de la izquierda los distintos hilos que se están ejecutando en el programa (aquí hay un total de 8) y, en la parte de la derecha, cómo ha llegado la ejecución hasta el punto de interrupción actual (en este caso: `main()` ha llamado a `run()`; la lista sería más larga si no se hubiéramos detenido en una función llamada por `run()`). En la parte de la izquierda podemos inspeccionar las variables locales, incluido el objeto actual (el objeto al que apunta la variable `this`).

Una vez aquí, disponemos de diversas posibilidades: puede ejecutar la línea actual (**F10**, como la orden 'next' de gdb), avanzar dentro de funciones (**F11**, como la orden 'step' de gdb) o ejecutar hasta el final de la función (**F12**, como la orden 'finish' de gdb). En cada paso, KDevelop actualiza las variables mostradas en la parte de la izquierda con sus valores actuales. También puede situar el ratón sobre cualquier símbolo del código fuente (por ejemplo, una variable) para que KDevelop muestre su valor actual y le ofrezca la posibilidad de detener la ejecución del programa la próxima vez que se modifique el valor de dicho símbolo. Si sabe manejar gdb, también puede pulsar el botón de la herramienta **GDB** que hay en la parte inferior para disponer de la posibilidad de introducir órdenes de gdb, como, por ejemplo, para cambiar el valor de una variable (para la que no exista otro modo de hacerlo).

## 7.2. Adjuntar el depurador a un proceso en ejecución



A veces, deseamos depurar un programa que ya está en ejecución. Un escenario para ello consiste en la depuración de programas en paralelo usando [MPI](#), o para depurar un proceso en segundo plano de larga ejecución. Para ello, vaya a la entrada del menú **Ejecutar** → **Adjuntar a proceso**, que abrirá una nueva ventana como la anterior. Entonces querrá seleccionar el programa que coincida con el proyecto actualmente abierto en KDevelop (en nuestro caso sería el programa `step-32`).

Esta lista de programas puede resultar confusa debido a que suele ser larga, como en el caso que se muestra aquí. Puede hacer que las cosas sean más simples usando la lista desplegable que hay en la parte superior derecha de la ventana. El valor por omisión es **Procesos del usuario**, es decir, todos los programas que ejecutan cualquiera de los usuarios que tienen iniciada sesión en esta máquina (si se trata de su equipo de escritorio o de su portátil, es probable que usted sea el único usuario, además del usuario «root» y de otras cuentas usadas por algunos servicios). No obstante, la lista no incluye los procesos ejecutados por el usuario «root». Puede limitar la lista seleccionando **Procesos propios**, o bien eliminando todos los programas ejecutados por otros usuarios. E, incluso mejor, seleccionando **Solo programas**, que eliminará gran cantidad de procesos que se ejecutan bajo su nombre de usuario pero con los que raramente interactuará (como el gestor de ventanas), tareas en segundo plano y similares, que no serán candidatos para depurar.

Una vez haya seleccionado un proceso, adjuntar a él le llevará al modo de depuración de KDevelop, abrirá todas las vistas de herramientas típicas de depuración y detendrá el programa en la posición en la que lo adjuntó a él. En ese caso tal vez quiera fijar puntos de interrupción, puntos de vista o lo que necesite y continuar la ejecución de programa yendo a la opción del menú **Ejecutar** → **Continue**.

## 7.3. Algunos atajos de teclado útiles

## Manual de KDevelop

<b>Depuración</b>	
<b>F10</b>	Avanzar sobre ('siguiente' de gdb)
<b>F11</b>	Avanzar dentro ('paso' de gdb)
<b>F12</b>	Avanzar fuera ('finalizar' de gdb)

## Capítulo 8

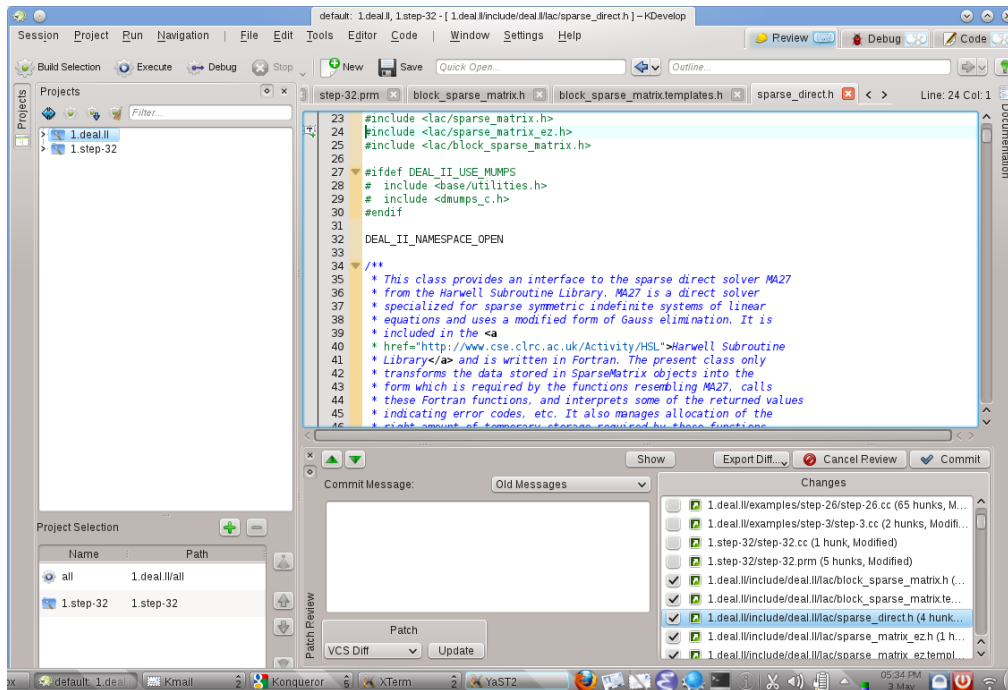
# Trabajar con sistemas de control de versiones

Si está trabajando con proyectos grandes, es posible que el código fuente esté gestionado por un sistema de control de versiones, como [subversion](#) o [git](#). La siguiente descripción se ha escrito teniendo en cuenta **subversion**, pero será igualmente válida si utiliza **git** o cualquier otro sistema de control de versiones implementado.

Para empezar, si el directorio en el que se encuentra situado el proyecto está bajo control de versiones, KDevelop se dará cuenta automáticamente. En otras palabras: no es necesario que le indique a KDevelop que descargue una copia por sí mismo cuando configure el proyecto; basta con hacer que KDevelop apunte a un directorio en el que haya descargado previamente una copia del repositorio. Si tiene un directorio bajo control de versiones de la forma indicada, abra la vista de la herramienta **Proyectos**. En ese momento podrá hacer varias cosas:

- Si el directorio está desactualizado, puede actualizarlo a partir del repositorio: pulse en el nombre del proyecto con el botón derecho del ratón, vaya al menú **Subversion** y seleccione **Actualizar**. Esto hará que todos los archivos que pertenezcan al proyecto se actualicen con respecto al repositorio.
- Si desea restringir esta acción a subdirectorios o archivos individuales, expanda la vista de árbol de este proyecto hasta el nivel necesario y haga clic con el botón derecho del ratón en el nombre de un subdirectorio o de un archivo y haga luego lo mismo que antes.

## Manual de KDevelop



- Si ha editado uno o más archivos, expanda la vista del proyecto hasta el directorio en el que están ubicados dichos archivos y pulse con el botón derecho del ratón sobre el directorio en cuestión. Esto le proporcionará un menú **Subversion** que le ofrece distintas opciones. Elija **Comparar con la base** para ver las diferencias entre la versión que haya editado y la versión del repositorio que hubiera descargado previamente para actualizar su copia local (la revisión «base»). La vista resultante le mostrará las diferencias existentes en todos los archivos del mencionado directorio.
- Si solamente ha editado un archivo, también puede obtener el menú **Subversion** para dicho archivo pulsando con el botón derecho del ratón sobre su nombre en la vista del proyecto. Todavía más sencillo, pulse con el botón derecho del ratón sobre la vista del **Editor** en la que esté abierto el mencionado archivo para obtener la misma opción de menú.
- Si quiere revisar uno o más archivos modificados, pulse con el botón derecho del ratón sobre un archivo individual, sobre un directorio o sobre un proyecto completo y seleccione **Subversion** → **Enviar**. Esto le llevará al modo de **Revisión**, el tercer modo además de **Código** y **Depuración**, como puede ver en la esquina superior derecha de la ventana principal de KDevelop. La imagen de la derecha se lo muestra. En el modo de **Revisión**, la parte superior le muestra las diferencias para todo el subdirectorio o proyecto y cada archivo individual con las modificaciones que contiene resaltadas (vea las distintas pestañas de esta parte de la ventana). Por omisión, todos los archivos modificados pertenecen al conjunto de cambios que está a punto de enviar, aunque puede quitar algunos de ellos de la selección si considera que sus modificaciones no están relacionadas con lo que desea enviar. Por ejemplo, como se muestra a la derecha, hemos desmarcado `step-32.cc` y `step-32.prm` porque los cambios que contienen estos archivos no tienen nada que ver con el resto de modificaciones de este proyecto y no queremos enviarlos al repositorio todavía (porque vamos a hacerlo más adelante en otro envío). Tras revisar los cambios, puede crear un mensaje descriptivo del envío en el cuadro de texto y pulsar el botón **Enviar** que hay a la derecha para entregar los cambios.
- Cuando está visualizando las diferencias, puede entregar un único archivo pulsando con el botón derecho del ratón en la ventana del editor para obtener la opción del menú **Subversion** → **Enviar**.

## Capítulo 9

# Personalización de KDevelop

Existen ocasiones en las que deseará cambiar el aspecto o el comportamiento por omisión de KDevelop como, por ejemplo, cuando está acostumbrado a distintos atajos de teclado o cuando su proyecto necesita un estilo de sangrado distinto para el código fuente. En las siguientes secciones hablaremos brevemente sobre los diferentes modos de personalizar KDevelop para estos propósitos.

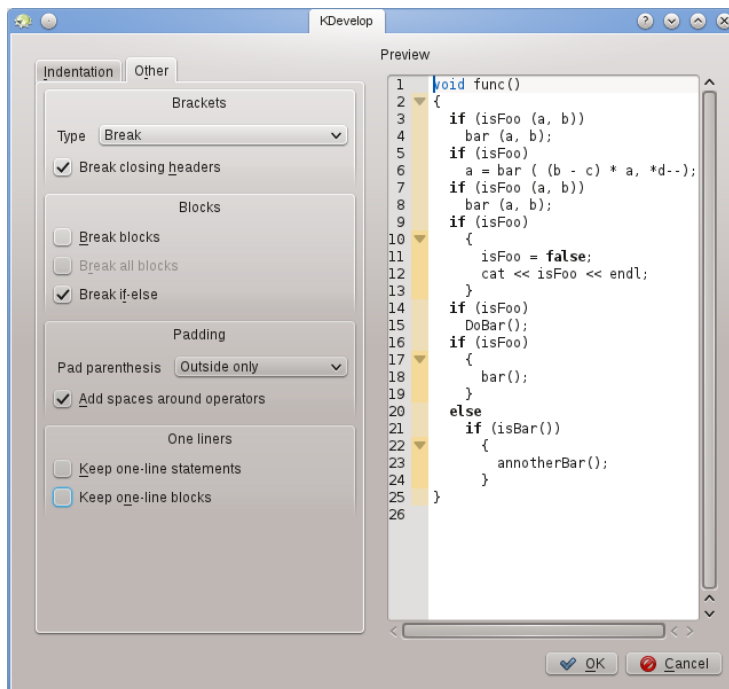
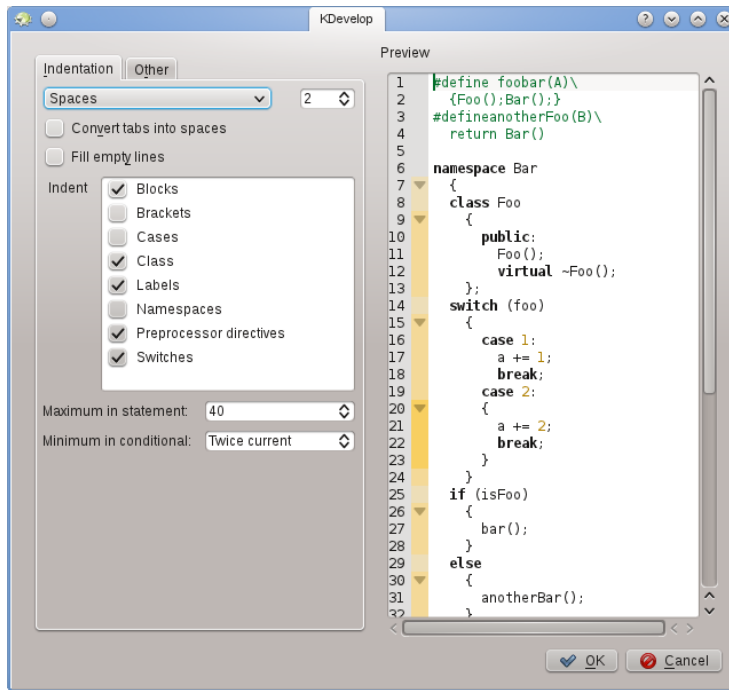
### 9.1. Personalización del editor

Existen varias cosas útiles que se pueden configurar relacionadas con el editor integrado de KDevelop. Una de las más usuales consiste en mostrar la numeración de las líneas usando la opción del menú **Editor** → **Ver** → **Mostrar números de líneas**, lo que facilita la identificación de las líneas de código que se muestran en los mensajes de error o de depuración. En el mismo submenú también puede activar el *borde de iconos* (una columna a la izquierda del código en la que KDevelop mostrará distintos iconos, como cuando fija un punto de interrupción en la línea actual).

### 9.2. Personalización de la sangría del código

A muchos de nosotros nos gusta tener el código fuente formateado de un modo particular. Muchos proyectos pueden forzar un estilo de sangría determinado. Es posible que ninguno de ellos coincida con el estilo por omisión de sangría que utiliza KDevelop. No obstante, puede personalizar este comportamiento usando la opción del menú **Preferencias** → **Personalizar KDevelop** y seleccionando en la parte de la izquierda **Formateador de código fuente**. Puede seleccionar cualquiera de los estilos de sangría predefinidos y más ampliamente usados, o bien definir el suyo propio añadiendo un nuevo estilo y editándolo. Es posible que no exista un modo exacto de volver a crear el estilo de sangría que usaban sus proyectos en el pasado, pero puede aproximarse bastante usando los ajustes de un nuevo estilo. Las dos imágenes inferiores muestran un ejemplo de ello.

# Manual de KDevelop



**NOTA**

En **KDevelop 4.2.2** puede crear un nuevo estilo para un tipo MIME determinado (por ejemplo, para los archivos de cabecera de C++), aunque dicho estilo no se mostrará en la lista de los posibles estilos para otros tipos MIME (por ejemplo, para los archivos de código fuente de C++). Por supuesto, es posible que este nuevo estilo sea apropiado para ambos tipos de archivos. En este caso deberá definir el estilo dos veces, uno para los archivos de cabecera y otro para los de código fuente. Este comportamiento se ha notificado como el [error 272335 de KDevelop](#).

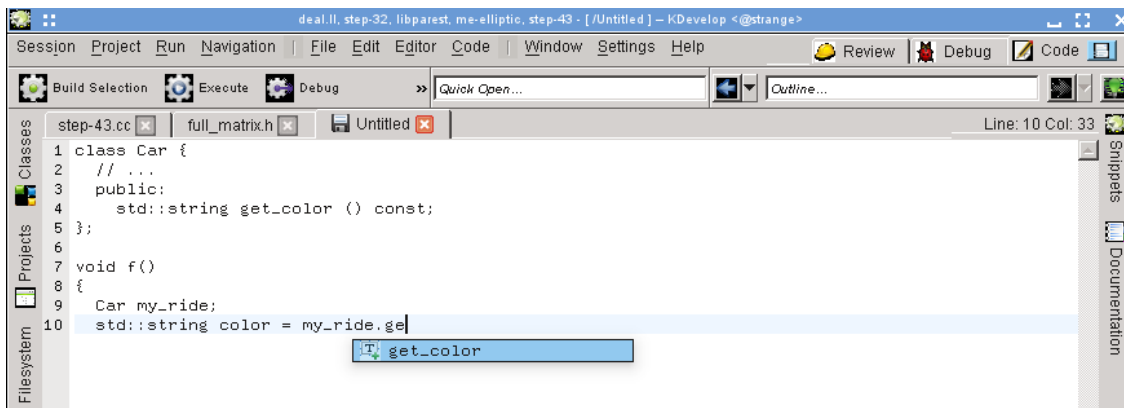
### 9.3. Personalización de los atajos de teclado

KDevelop dispone de una lista casi ilimitada de atajos de teclado (algunos de ellos se listan en las ‘útiles secciones sobre atajos de teclado’ de diversos capítulos de este manual) que se pueden modificar de acuerdo a sus gustos usando la opción **Preferencias** → **Configurar los atajos de teclado** del menú. En la parte superior del diálogo puede introducir una palabra a buscar para que únicamente se muestren las órdenes que coincidan; entonces podrá editar la combinación de teclas que esté asociada con dicha orden.

Uno que se ha encontrado muy útil para modificar es el que se usa para fijar la **Alineación** de la tecla **Tab** (muchas personas no suelen introducir tabuladores a mano y prefieren que el editor elija la distribución del código fuente; con el atajo de teclado cambiado, al pulsar **Tab**, KDevelop aumenta o disminuye la alineación del código fuente). Otro de ellos es asignar **Cambiar punto de interrupción** a **Ctrl-B**, ya que se trata de una operación muy frecuente.

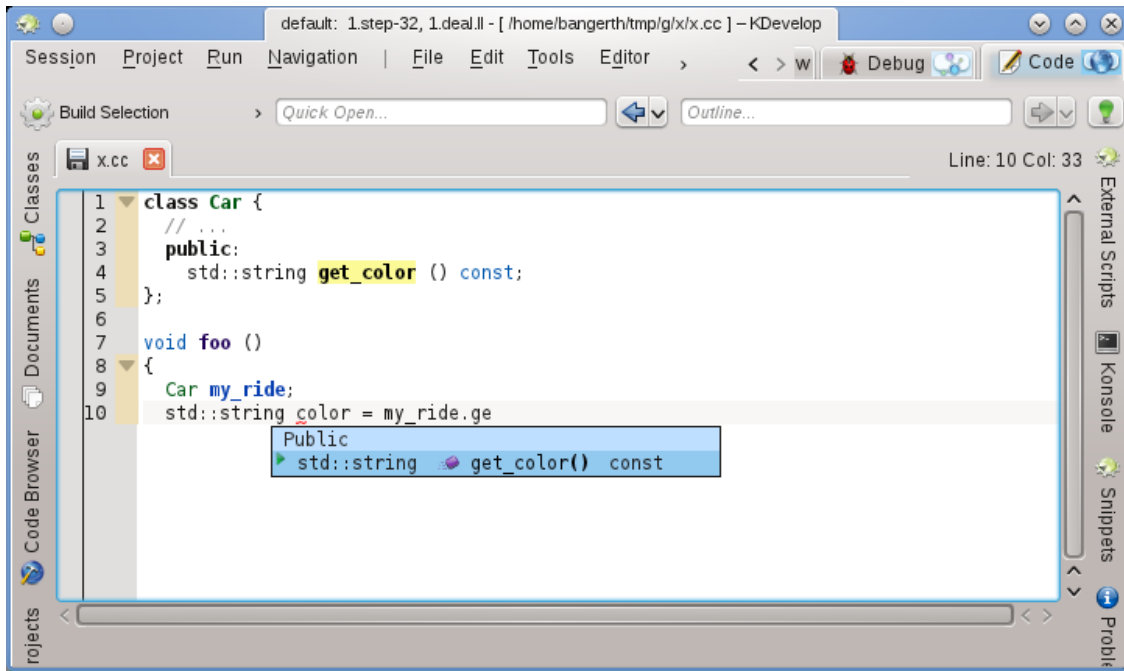
### 9.4. Personalización de la terminación automática de código

La terminación de código se describe en [esta sección del manual sobre la escritura de código fuente](#). En KDevelop, proviene de dos fuentes: el editor y el motor de análisis sintáctico. El editor (Kate) es un componente del entorno KDE que ofrece terminación automática basada en palabras que ya ha visto en otras partes del mismo documento. Esta terminación se puede identificar en la ayuda emergente por el icono que la precede:



La terminación de código del editor se puede personalizar usando **Preferencias** → **Configurar el editor** → **Edición** → **Terminación automática**. En particular, puede seleccionar cuántos caracteres de una palabra necesita teclear antes de que la terminación automática entre en funcionamiento.

Por otra parte, la terminación automática propia de KDevelop es mucho más potente, ya que tiene en cuenta información semántica sobre el contexto. Por ejemplo, sabe qué funciones miembro debe ofrecer cuando escribe `object.`, etc., como se muestra a continuación:



Esta información de contexto proviene de diversos complementos de implementaciones de lenguajes, que se pueden usar tras guardar un archivo determinado (para que se pueda comprobar el tipo de archivo y usar la implementación de lenguaje correcta).

La terminación de KDevelop está preparada para mostrarse cuando teclea, en seguida, casi en cualquier lugar donde se pueda terminar alguna cosa. Esto se puede configurar en **Preferencias** → **Configurar KDevelop** → **Implementación de lenguaje**. Si no se ha activado con anterioridad (como debería, por omisión), asegúrese de que ha marcado **Activar invocación automática**.

KDevelop tiene dos modos de mostrar una terminación: la **terminación automática mínima** muestra solamente la información básica en ayudas emergentes (es decir, el nombre del espacio de nombres, la clase, la función o la variable). Esto se mostrará de forma similar a la terminación de Kate (a excepción de los iconos).

Por otra parte, la **terminación completa** también le mostrará el tipo de cada entrada y, en el caso de funciones, los argumentos que posee. Además, si está rellenando los argumentos de una función, la terminación completa le mostrará un cuadro adicional de información por encima del cursor que le mostrará el argumento actual sobre el que esté trabajando.

La terminación de código de KDevelop también debe llevar al primer lugar y resaltar en verde cualquier elemento de terminación que coincida con el tipo esperado en la actualidad tanto en la terminación mínima como en la completa, lo que se conoce como las 'mejores coincidencias'.

Las tres elecciones posibles para el nivel de terminación del diálogo de configuración son:

- **Siempre terminación mínima:** no mostrar la 'Terminación completa'
- **Terminación automática mínima:** mostrar solo la 'Terminación completa' cuando la terminación automática se haya lanzado de forma manual (es decir, cada vez que pulse la combinación **Ctrl-Espacio**)
- **Siempre terminación completa:** mostrar siempre la 'Terminación completa'

## Capítulo 10

# Créditos y licencia

Para el copyright de la documentación, vea el [historial de la página de KDevelop4/Manual](#)

Traducido por Eloy Cuadra [ecuadra@eloihr.net](mailto:ecuadra@eloihr.net).

Esta documentación está sujeta a los términos de la [Licencia de Documentación Libre GNU](#).