wxWidgets GUI-Programmierung: Text File Viewer / Editor

Klaus Kusche

Wir wollen ein einfaches wxWidgets-Programm schreiben,
mit dem man **Textfiles anschauen** kann:
Links im Fenster soll eine **Verzeichnis-Baumansicht** sichtbar sein
und rechts soll der **Inhalt** des links doppelgeklickten Files angezeigt werden.

Im Detail (am besten ausgehend von unserem wxWidgets-Hello-Beispiel):

- Unsere *Applikationsklasse* und der Code ihrer *Init-Methode*
  schauen genauso aus wie in unserem Hello-Beispielprogramm aus der Vorlesung.
- Unsere *Hauptframe-Klasse* hat neben dem Konstruktor *einen Event-Handler*
  für den Doppelklick eines Files im Verzeichnis-Baum
  (wobei das Argument kein **`wxCommandEvent&`** sondern ein **`wxTreeEvent&`** ist).

  Weiters hat unsere Hauptframe-Klasse *3 Member* (jeweils als *Pointer*!):

  - Ein **`wxSplitterWindow`**: Es teilt ein Fenster in zwei Subfenster.
  - Ein **`wxGenericDirCtrl`**: Das ist der Verzeichnis-Baum links.
  - Ein **`wxTextCtrl`**: Das ist das Textfeld rechts.
- *Event-Nummern* brauchen wir keine:
  Wir behandeln nur einen einzigen Event einer einzigen Quelle und müssen daher
  nicht zwischen verschiedenen Buttons oder Menü-Einträgen unterscheiden
  (im **`Bind`** kann man in diesem Fall den *vierten Parameter* einfach *weglassen*).
- Der *Konstruktor* unseres Hauptframes hat zuerst einmal
  4 Einträge in der *Initialisierungsliste*:
  - Als Erstes wird so wie im Hello-Programm der *Parent-Konstruktor*
    aufgerufen (wobei man die Größe und die Position weglassen kann:
    Diese haben sinnvolle Defaults.)
  - Dann kommen die *Initialisierungen der drei Pointer-Member*
    mit einem entsprechenden **`new`** samt *Konstruktor-Aufruf*:
    - Das erste Argument ist immer das *Vater-Fenster in der Anordnungs-Schachtelung*, also **`this`** (das Hauptframe) beim Splitter
      und der *Splitter* bei Verzeichnis-Baum und Textfeld.
    - Das zweite Argument (die Id) ist in unserem Fall immer **`-1`**.
    - Beim *Text* kommen dann noch der *Leerstring* als initialer Inhalt,
      `wxDefaultPosition` und **`wxDefaultSize`** sowie die *Style-Flags*
      für Readonly, Multiline, horizontaler Scrollbar statt Zeilenumbruch
      und normaler Eingabe von Tabs (*siehe Doku:* Klasse **`wxTextCtrl`**,
      Kapitel "Styles", die einzelnen Bit-Konstanten werden mit **`|`**
      (bitweisem Oder) verbunden!).

- Im *Body des Konstruktors* passiert Folgendes (such dir die Methoden aus der Doku!):
    - Wir müssen dem Splitter sagen, dass er sich *vertikal teilen* soll
      (mit einer vernünftigen Position, z.B. 150 Pixel von links) und
      dass *links unser Directoy-Tree und rechts unser Text* angezeigt werden soll.

      Weiters habe ich im Splitter noch eine *minimale Teilfenster-Breite* eingestellt,
      damit man den Trennstrich nicht so weit an den Rand verschieben kann,
      dass er unsichtbar wird.
    - Dem Tree müssen wir sagen, *welches Verzeichnis* er vorselektieren soll.
      Gib als Verzeichnis **`wxGetCwd()`** an
      (das ist eine globale Funktion, sie liefert das aktuelle Verzeichnis).
    - Zuletzt müssen wir den Event **`wxEVT_COMMAND_TREE_ITEM_ACTIVATED`**
      mit unserem *Event-Handler* verbinden
      (die Event-Id der Quelle dieses Events ist uns egal).
- Der *Event-Handler* selbst ist ganz einfach:
  Wir fragen den Tree nach dem Filenamen des gerade selektierten Eintrags und
  sagen dem Textfeld, dass es diesen File laden soll (für beides gibt es Methoden).
  Du könntest die Tree-Methode verwenden, die *nur für Files* Namen liefert
  und für Klicks auf Dir's den Leerstring, den könntest du mit einem **`if`** abfangen.

Erweiterung 1: Buttons und Sizer

Damit wir auch *Buttons* (Klasse **`wxButton`**) und *Sizer* üben,
geben wir unserem Fenster *unten 4 Buttons nebeneinander*:

- Einen Exit-Button zum Beenden.
- Drei Buttons zum Umsschalten der Anzeige
  zwischen nur Tree, nur Text oder beidem.

Wir brauchen also *vier Event-Handler* (der Event-Typ von Buttonklicks ist
`wxCommandEvent`), die wir in unserer Hauptframe-Klasse deklarieren müssen.
Weiters brauchen wir *vier Event-Nummern* und *vier* **`Bind`**-*Aufrufe* im Konstruktor
für **`wxEVT_COMMAND_BUTTON_CLICKED`** Events mit den jeweiligen Id's.

- Der *Exit-Event-Handler* ruft wie im Hello-Beispielprogramm nur **`Close`** auf.
- Bei den anderen drei Handlern muss man *jeweils 3 Fälle unterscheiden*:
  Ist der Splitter gerade geteilt (das bekommt man mit **`IsSplit`** heraus),
  und (falls es nicht geteilt ist) welches Teilfenster wird gerade angezeigt
  (das aktuell angezeigte Teilfenster bekommt man mit **`GetWindow1`**,
  man vergleicht den resultierenden Pointer mit dem Verzeichnisbaum-Member
  bzw. dem Text-Member unserer Hauptframe-Klasse):
    - Will man von *geteilt auf ungeteilt* umschalten, muss man für den Splitter
      `Unsplit` aufrufen mit dem Teilfenster, das verschwinden soll.
    - Will man von *ungeteilt auf geteilt* wechseln, muss man **`SplitVertically`**
      mit beiden Teilfenstern aufrufen und danach für beide Teilfenster **`Show`**.
    - Will man das schon *ungeteilte Fenster wechseln*, verwendet man
      die Splitter-Methode **`ReplaceWindow`**. Außerdem muss man
      auf das alte Teilfenster **`Hide`** und auf das neue **`Show`** anwenden.
    - Stimmt die Anzeige ohnehin schon, sollte man *nichts* tun!

Bleibt noch das *Anlegen der Buttons* und die *geometrische Aufteilung des Hauptfensters* mit Sizern im Konstruktor des Hauptframes:

- Wir brauchen für die Buttons und Sizer *keine* Member-Variablen, es reichen *lokale Pointer im Konstruktor*.
- Zuerst legen wir *die vier Buttons* mit **new** an.
  Vaterfenster ist **this** (unser Hauptframe), Id ist die zum Button gehörige Event-Id, und die *Beschriftung* kann ein **&** für den *Shortkey* enthalten.
- Dann legen wir mit **new** *zwei Sizer* an, und zwar **wxBoxSizer**:
  Einen *horizontalen* für die vier Buttons und
  einen *vertikalen*, der das Hauptfenster in den Splitter und die vier Buttons teilt.
- Genau das ist auch unser nächster Schritt:
  Wir fügen *die vier Buttons zum horizontalen Sizer* hinzu
  und dann *den Splitter und den horizontalen Sizer zum vertikalen Sizer*.

  Ich habe jeweils die **Add**-Variante mit Window, Proportion (bei den Buttons für alle 1, beim Hauptsizer 1 für den Splitter und 0 für die Buttons, die ja nicht wachsen sollen) und Flags (Doku lesen und Effekt probieren!) sowie nur bei den Buttons zusätzlich 5 Pixel Rand aufgerufen. Achtung: Das steht nicht in der Doku von **wxBoxSizer**, sondern in der Vaterklasse **wxSizer**!
- Dann habe ich dem vertikalen Sizer noch eine *Minimal-Größe* gegeben.
- Als letzten Schritt muss man den *vertikalen Sizer und das Haupt-Frame gegenseitig miteinander verbinden*: Man ruft dafür **SetSizerAndFit** des Hauptframes auf.

Erweiterung 2: Editier-Funktion

Wir wollen aus dem Viewer einen *Editor* machen:

- Dazu müssen wir uns im Event-Handler für das Anzeigen eines Files den aktuellen *Filenamen* merken (in einer *neuen Member-Variablen* vom Typ **wxString**, auf den Leerstring initialisieren!) und das Textfeld mit **SetEditable** schreibbar machen.
- Weiters schreiben wir in unserem Hauptframe eine *neue Hilfs-Methode*, die den Inhalt des Textfeldes (nur falls er *modifiziert* ist, das Textfeld sagt uns das!) mit **SaveFile** in den in unserem Member gemerkten Filenamen *zurückschreibt*.

  Als Zusatzaufgabe bekommt die Methode einen **bool**-Parameter, der festlegt, ob ungefragt gespeichert oder zuerst mit einer Ja-Nein-Messagebox rückgefragt werden soll (die globale Funktion **wxMessageBox** verwenden!).
- Weiters kommt ein *fünfter Button zum Speichern* dazu, dessen Event-Handler diese Hilfs-Methode aufruft.
- Auch der bestehende Event-Handler für das *Lesen eines neuen Files* muss zuerst die Hilfs-Methode zum *Speichern des alten Files* aufrufen (mit Rückfrage)!
- Ganz perfekt wäre es, das auch Default-Handling von **Close** (Exit-Button oder das **X** oben rechts am Fensterrahmen) zu *überschreiben*, um zu speichern.
  Dazu braucht unsere Hauptframe-Klasse eine *Close-Event-Handler-Methode* (Parameter **wxCloseEvent&**, **Bind** ohne Id auf **wxEVT_CLOSE_WINDOW**), der zuerst unsere Speichern-Hilfsmethode und dann **Destroy()** aufruft.