15. Einführung in die GUI-Programmierung[1]

Bislang liefen sämtliche MATLAB-Anwendungen über das Kommandofenster: das Programm wurde gestartet mit einer Anweisung in eben diesem Fenster (durch Angabe des Files, in dem sich das Programm befindet), Eingaben wurden dort hineingetippt und ebenso Ausgabe dort dargestellt. Eine kleine Abweichung stellt das Vorgehen des Lesens aus und Schreibens in eine (externe) Datei dar.

Wir werden eine weitere Variante kennenlernen, nämlich die Ein- und Ausgabe über Ein- und Ausgabefeldern in sogenannten Masken – und das Aufrufen und Steuern von Aktionen mit Hilfe ebensolcher Masken. Was nun Masken sind, wie diese aufgerufen werden – und insbesondere, was damit erreicht wird, ist Gegenstand dieses Kapitels.

Wir betrachten ohne grandiose Begriffsdefinition ein einführendes Beispiel:

```
% Kap15_01_DialogBox_Einf

% Frage nach Programmabbruch durch Dialogbox
durchlauf = 'j';
while durchlauf == 'j'
   disp('Programmdurchlauf');
   button = questdlg('Noch einen Programmdurchlauf?', 'Abbruch',...
                  'ja','nein', 'ja');
   switch button
       case 'ja'
          durchlauf = 'j';
       case 'nein'
          durchlauf = 'n';
   end;
end
```

Hier ist also der unbekannte Befehl `button = questdlg(Parameterliste)` zu sehen. Was er bewirkt, ersieht man am deutlichsten, wenn man nach alter Weise im Kommandofenster den Befehl zum Starten eingibt:

```
>> Kap15_01_DialogBox_Einf
```

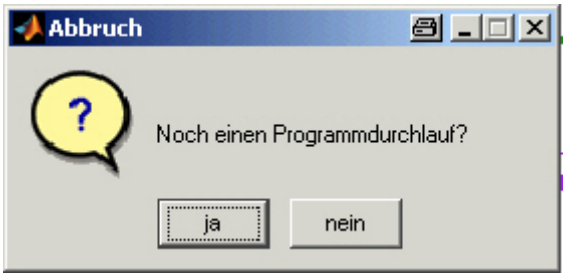
Es erscheint danach im Kommandofenster die Ausgabe:

```
Programmdurchlauf
>>
```

Bis hierher nichts Neues.

Gleichzeitig erscheint auf dem Bildschirm ein neues Fenster in Form eines Fragekastens:

[1] Dieses Kapitel folgt im wesentlichen dem dritten Kapitel „GUI“ aus dem Buch „Einstieg in das Programmieren mit MATLAB“ von U.Stein



Klickt man mit der Maustaste auf den Knopf „ja“ oder betätigt einfach nur die Enter-Taste, erscheint im Kommandofenster erneut die Ausgabe:

```
Programmdurchlauf
```

so daß der Inhalt des Fensters jetzt folgendes Aussehen hat:

```
>> Kap15_01_DialogBox_Einf
Programmdurchlauf
Programmdurchlauf
>>
```

Und besagtes Fenster erscheint erneut.

Klickt man diesmal den Knopf „nein“ an, verschwindet nicht nur das Fenster, sondern das Programm wird offensichtlich beendet.

Offensichtlich wird die Steuerung auf einen erneuten Programmdurchlauf über dieses Fenster vorgenommen.

Erstellen einer Dialog-Box zur Erstellung eines Abfrage-Dialogs:

Ein derartiges Fenster ist eine sogenannte Dialog-Box: diese stellen Standard-Dialoge zur Verfügung (wie man sie schon aus Windows kennt), die durch Anklicken mit der Maus auf sogenannte Knöpfe den weiteren Programmverlauf steuern.

Eine solche Dialog-Box für Abfragen wird durch den Befehl erzeugt, der folgendes Aussehen hat:

```
button =  questdlg('qstring','title','str1','str2','str3','default')
```

`qstring` ist der sogenannte String, der als Frage in der Mitte der Box placiert wird („question string“; also qstring = „Noch einen Programmdurchlauf?“)

`title` ist der String, der als Überschrift im Titelbalken (hier „Abbruch“)

`str1`, `str2`, `str3` sind die Aufschriften („Labels“) auf den Knöpfen. Es werden so viele Knöpfe angezeigt, wie derartige Strings angegeben sind (im Beispiel nur zwei, nämlich „ja“ und „nein“)

`default` enthält einen String, der schon als `str1` oder als `str2` oder einer der anderen der Knopfaufschriften vorhanden ist. Dadurch wird dieser Knopf als Standard-Knopf gesetzt; dies bedeutet, dass, wenn man nicht explizit einen Knopf mit der Maus ansteuert, sondern nur die Enter-Taste betätigt, dass eben der als „`default`“ gekennzeichnete Knopf aktiviert wird (hier also der Knopf „ja“).

Zurückgegeben wird für die Variable `button` der String, dessen dazugehöriger Knopf per Mausklick gerade aktiviert wurde.

Im Beispielprogramm `Kap15_01_DialogBox_Einf` wird die Abbruchvariable durchlauf also in Abhängigkeit des zurückgegebenen Wertes der besagten Box gesetzt:

```
   switch button
       case 'ja'
          durchlauf = 'j';
       case 'nein'
          durchlauf = 'n';
   end;
```

Damit wird das Verhalten erzeugt, dass wir bereits von den althergebrachten MATLAB-Programmen her gewöhnt sind.

Vor einem allgemeinen Einstieg noch eine kurze Variation des Programms, um mit den Parametern der Dialog-Box vertrauter zu werden:

```
% Kap15_02_DialogBox_Variation

% Frage nach Programmabbruch durch Dialogbox
durchlauf = 'j';
while durchlauf == 'j'
   disp('Programmdurchlauf');
   button = questdlg('Noch einen Programmdurchlauf?', 'Abbruch',...
                  'ja','nein', 'nein');
   fprintf('Wert des Buttons ist %s\n', button);
   switch button
       case ''
           durchlauf = 'n';
       case 'ja'
          durchlauf = 'j';
       case 'nein'
          durchlauf = 'n';
   end;
end
```

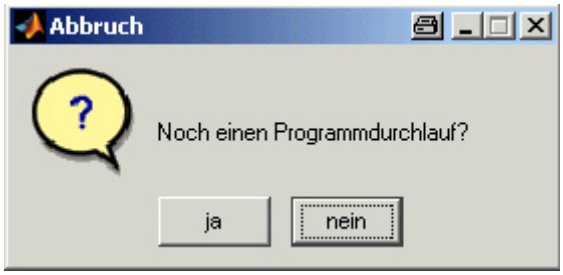
Gewaltige Änderungen wurden nicht vorgenommen:
Wir lassen auf den Bildschirm den jeweiligen Wert von `button` ausgeben, um zu erfahren, was passiert, wenn man das Symbol wählt „Maske schließen“.

Außerdem wurde als Standardknopf jetzt „nein“ gesetzt.

Der Aufruf

```
>> Kap15_02_DialogBox_Variation
```

bringt also wieder eine Dialog-Box auf den Bildschirm, die das Aussehen hat:



Durch die gestrichelte Umrahmung wird angezeigt, welcher Knopf der Standardknopf ist, der bei Betätigen der Enter-Taste ausgewählt wird. In der Tat wird das Programm jetzt sofort beendet, wenn einfach nur die Enter-Taste gedrückt wird. Im Kommandofenster zeichnet sich die Abfolge ab:

```
>> Kap15_02_DialogBox_Variation
Programmdurchlauf
Wert des Buttons ist nein
```

Rufen wir erneut das Programm auf und drücken diesmal „ja“ mit der Maustaste. Wie zu erwarten, wird die alte Box geschlossen, es erscheint erneut die gleiche Dialog-Box, und im Kommandofenster steht die Folge:

```
>> Kap15_02_DialogBox_Variation
Programmdurchlauf
Wert des Buttons ist ja
```

Drücken wir als nächstes auf den Icon in Form eines kleinen Kreuzes rechts oben in der Menüzeile der Box, wird das Programm beendet und damit die Box geschlossen. Im Kommandofenster steht dann

```
>> Kap15_02_DialogBox_Variation
Programmdurchlauf
Wert des Buttons ist ja
Programmdurchlauf
Wert des Buttons ist
>>
```

Der Inhalt von `button` ist also leer.

Dies wurde im Programm dahingehend berücksichtigt, dass auch dann ein Programmabschluß zu erfolgen hat:

```
switch button
    case ''
        durchlauf = 'n';
    case 'ja'
```

```
        durchlauf = 'j';
    case 'nein'
        durchlauf = 'n';
end;
```

Damit haben wir einführend ein wesentliches Element in der sogenannten GUI-Programmierung kennengelernt: die Auswahlmöglichkeiten, mit der weitere Aktionen gesteuert werden.

Was ist überhaupt GUI-Programmierung?
GUI ist die Abkürzung von „graphical user interface“, also graphische Benutzeroberfläche. Der Anwender interagiert also nicht direkt mit dem Rechner durch Eintippen bzw. Auslesen von Werten, sondern mittels gewisser Fenster, die auf dem Bildschirm erscheinen und die Eingabe von Werten erlauben bzw. die Ausgabe von Ergebnissen realisieren.

In den alten Computerzeiten wurden Interaktionen mit dem Rechner durch Eintippen mehr oder weniger kryptischer Befehle und langer Zahlenkolonnen auf einem schwarzen Bildschirm mit grünen (oder gelben) Buchstaben vorgenommen. Dann wartete man einen Moment – und der Rechner – wenn denn alles gutgegangen war - warf einem die Zahlenkolonnen auf den Bildschirm, in eine Datei (und damit möglicherweise auf einen Zeilendrucker). Das Kommandofenster in MATLAB ist wie die Shell in Linux oder die Windows-Eingabeaufforderung noch an diese spartanischen Zeiten angelehnt: Programmieren sollten eh nur Profis; Uneingeweihte haben schließlich am Bildschirm nichts verloren.

Die Idee, Aktionen und Abfolgen mittels verschiedener Programme zu visualisieren und damit durchschaubarer zu machen, haben nicht umsonst dazu beigetragen, dass der PC so weit verbreitet ist. Der Anwender gibt nicht mehr eine Folge skurriler Befehle ein (ohne sich dabei zu vertippen) – die voluminösen Handbücher immer in Reichweite -, sondern klickt Einträge in diverse Fenster an, die er wiederum aus einem Hauptfenster heraus durch Auswahl eines Menüs geöffnet hat.

All diese Fenster sind GUI’s – graphische Benutzeroberflächen.
Genaugenommen ist auch das Kommandofenster in MATLAB ein solches GUI – allerdings sowohl vom Aussehen als auch Funktionalität her so gehalten wie die alten Ein- und Ausgaben aus dem Computer-Neolithikum.

Ein solches GUI ist die Dialog-Box, von der wir bereits eine Form kennengelernt haben. Der Anwender kann sich hier nicht grandios vertippen – er hat ja nur zwei Knöpfe zum Anklicken zur Auswahl. Und selbst wenn er die Box einfach schließt oder die Enter-Taste betätigt, werden bestimmte Aktionen ausgelöst.

Callbacks

Was löst denn die Aktionen aus?
Die Antwort ist einfach: ein Mausklick oder eine Eingabe auf der Tastatur (Enter-Taste).
Es wird also das MATLAB-Programm gestartet.

Es werden möglicherweise bereits Werte erzeugt, die nachher mittels eines GUI's auf den Bildschirm gebracht werden sollen.

Im MATLAB-Programm steht der Befehl zum Öffne eines GUI's, z.B. mit dem Funktionsaufruf `questdlg`.

Die Dialog-Box (oder ein anderer GUI) wird nun auf den Bildschirm gebracht, wobei die Initialisierungswerte aus dem MATLAB-Programm verwendet werden. Durch den Funktionsaufruf wurde ein weiteres, (für den Anwender) unsichtbares Programm gestartet; dieses regelt die Größe des GUI, dessen Inhalt und Placierung auf dem Bildschirm.

Ab dann dreht das GUI-Programm Däumchen, bis es dem geneigten Anwender gefällt, zur Aktion zu schreiten. Das besagte Programm im Hintergrund enthält nämlich eine `while`-Schleife, die für den Benutzer unsichtbar so lange durchlaufen wird, bis letztendlich einer vom werten Benutzer ausgelöste Aktion das Betriebssystem veranlasst, eine Meldung zu schicken, die als Abbruchkriterium der Schleife dient.

Dem Windows-Betriebssystem wurde gemeldet, dass ein neues Fenster geöffnet wurde. Dies nimmt es zur Kenntnis, indem es (auch noch) dieses Fenster registriert.

Das GUI-Programm, das die Ausgabe des Fensters regelt, dreht derweil dank der `while`-Schleife weiter Däumchen (allerdings ohne mögliche andere Programme zu blockieren).

Diese `while`-Schleife wird erst verlassen (bzw. das GUI-Programm fortgesetzt), wenn vom Betriebssystem die entsprechende Anweisung kommt.

Wo bekommt das Betriebssystem die Anweisung her?
Das Betriebssystem regelt (unter anderem) die Fensterverwaltung: welches Fenster ist das aktive (und überdeckt die anderen), welche Aktionen werden über die Fenster ausgelöst.

Der Anwender starrt also weiterhin auf die Dialog-Box und entschließt sich zu einer Aktion. Aktion heißt hier: Betätigung einer Taste (Enter-Taste, Tab-Taste) oder der Maus oder Änderung der Mausposition.

Diese Aktion bekommt das Windows mit; es weiß, welches das aktive Fenster ist, also teilt es dem wartenden GUI-Programm mit: Knopf so und so wurde angeklickt, Taste so und so wurde betätigt (z.B. kann man mit der Tabulator-Taste zwischen den Knöpfen hin- und her wandern).

Das GUI-Programm schreckt aus seiner kontemplativen Phase und gibt den Wert an das aufrufende MATLAB-Programm weiter. Das sollte dann wissen, wie es darauf reagieren sollte.

Also:
Das große Warten auf den Befehl, der vom Anwender gegeben wird, ist das große Geheimnis der GUI's: warte so lange, bis der Rückruf von Betriebssystem erfolgt, der sagt, wie weiter zu verfahren ist.
Dem Programmierer in MATLAB obliegt es dann, auf die Rückgabewerte zu reagieren, indem er dafür bestimmte Funktionen geschrieben hat. Diese Funktionen als Reaktion auf den **Rückruf des Betriebssystems** heißen „Rückruf-Funktionen" oder „Callback".

GUIDE
Das Hin- und Her der betriebsinternen Funktionen, die gegenseitigen Aufrufe und insbesondere die komplexen Abläufe bei Erstellung eines Fensters auf dem Bildschirm sind auch von erfahrenen Programmierern schwer durchschaubar. Glücklicherweise gibt es mächtige Werkzeuge, die einem (also Normalsterblichen) sehr viel Arbeit bei der Erstellung

eines GUI's abnehmen. Das zu MATLAB gehörige Tool zur Erzeugung und Verwaltung von GUI's heißt GUIDE („graphical user interface design environment").

Nachfolgend werden wir uns allmählich in die Welt dieser Entwicklungsumgebung einarbeiten.

GUIDE starten:

Eingabe des Befehls

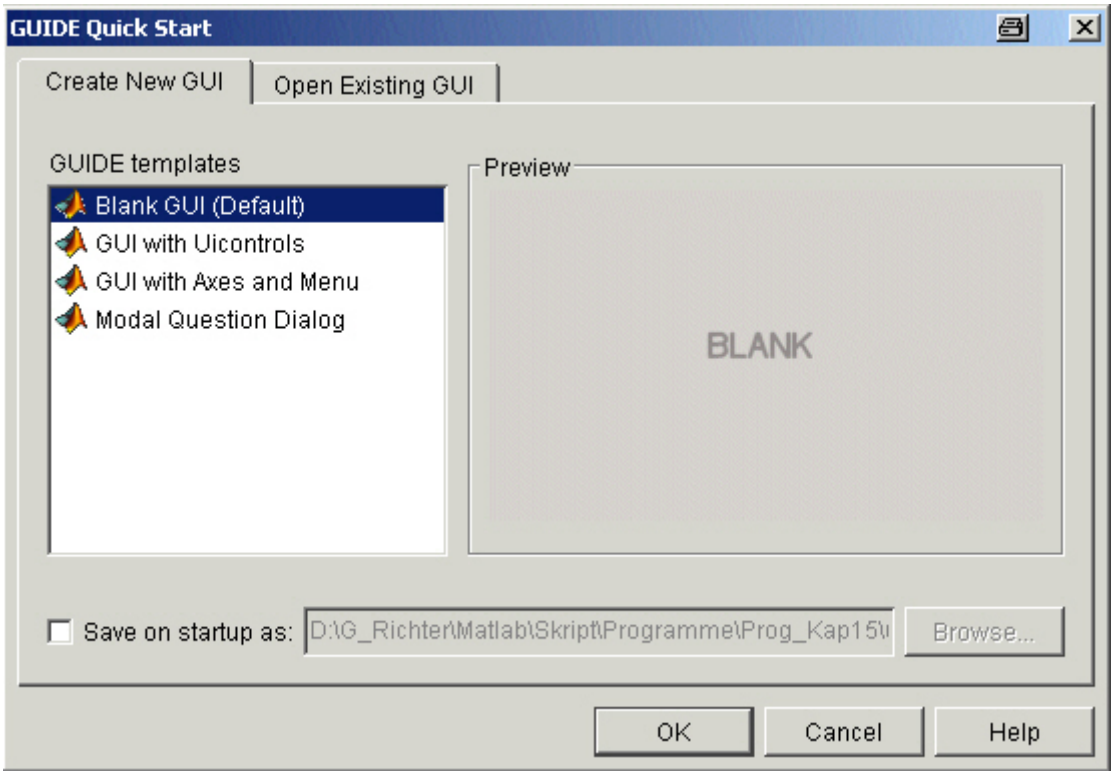
„`>> guide`" im Kommandofenster

oder in der Menüauswahl

„`File`" -> „`New`" -> „`GUI`"

Es öffnet sich der Startdialog von GUIDE

(normalerweise sollte das Fenster des Tabreiters „Create New GUI" geöffnet werden, doch notfalls kann dies durch einfaches Anklicken mit der Maus auf den entsprechenden Reiter erreicht werden, so dass sich folgendes Bild ergibt):



leeres GUI erzeugen

Dies ist nicht sonderlich schwer, denn es ist die Standardvorgabe, wie an der blau unterlegten ersten Zeile in der Auswahlliste zu ersehen.

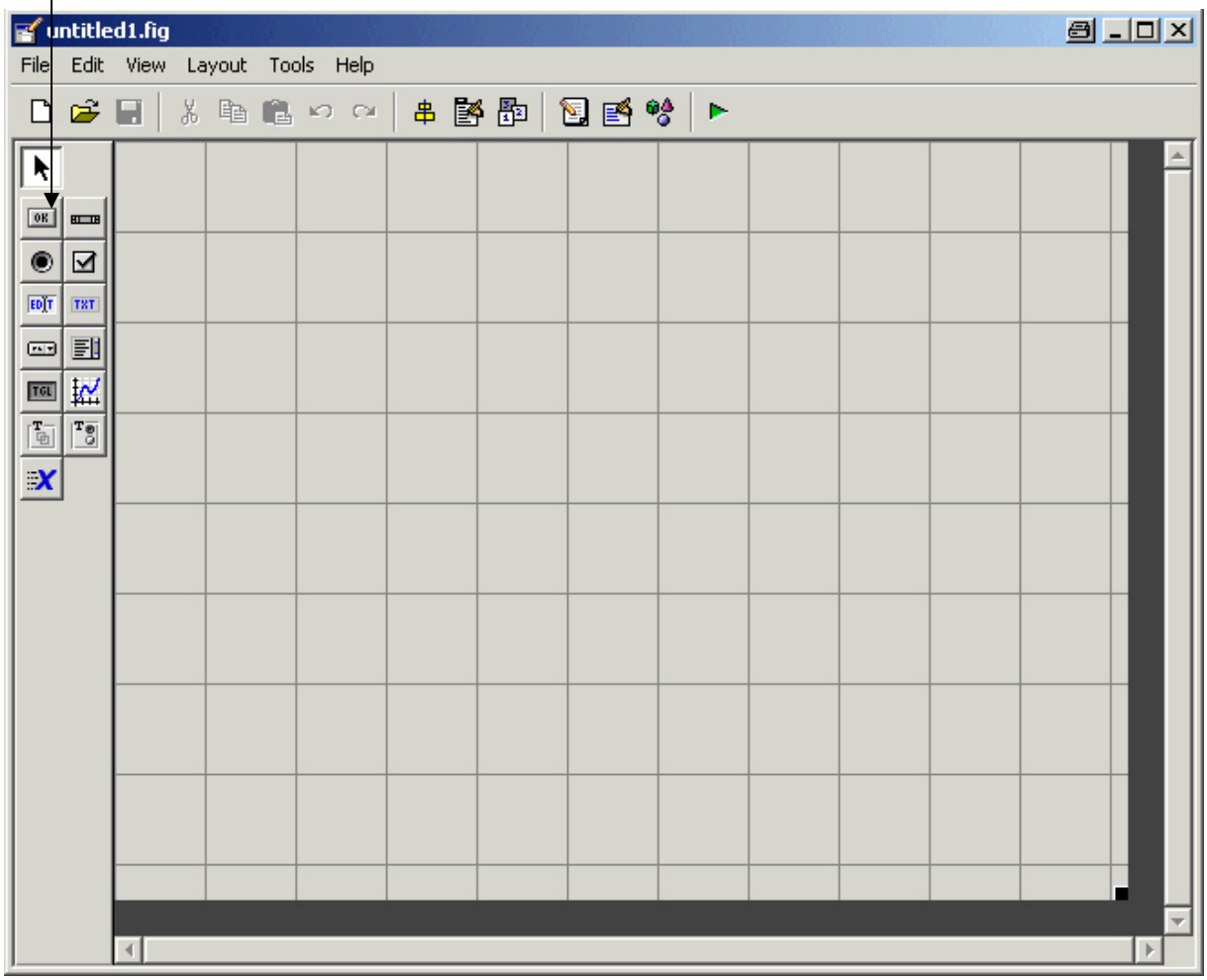
Enter-Taste oder Anklicken von „OK“ öffnet sich der GUIDE-Layout-Editor:

Rechts ist das leere Feld des GUI’s zu sehen. Das dort sichtbare Gitter soll die Positionierung der verschiedenen Elemente erleichtern.
Links befindet sich eine Auswahlpalette von Icons zur Auswahl, die dem unterschiedlichen Elemente in einem GUI darstellen.

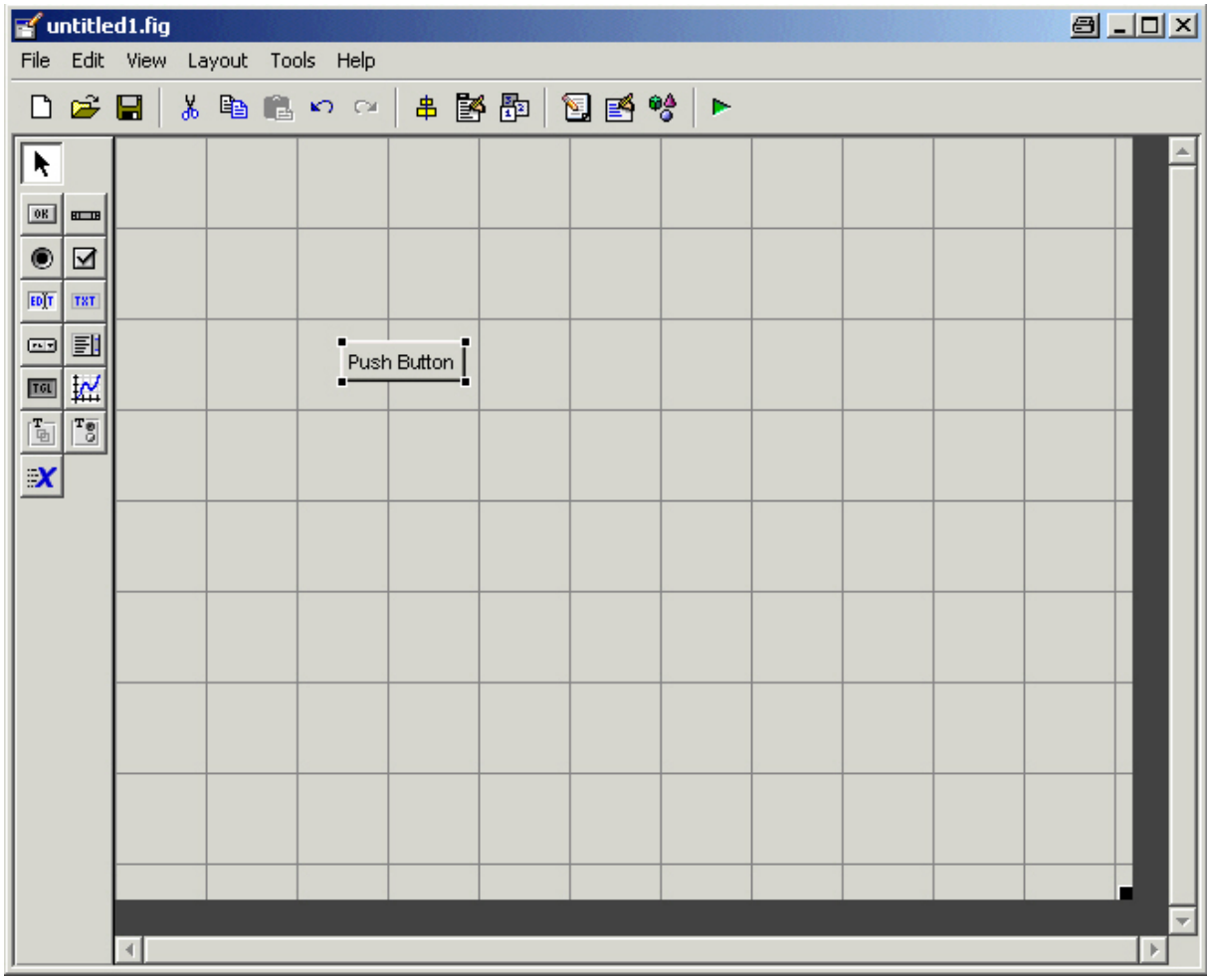
In diesem Beispiel soll nur ein einziges Element ausgewählt werden: ein Push-Button („Funktions-Knopf“ oder „Schaltfläche“). Dieser, wenn mit der Maus angeklickt, soll eine bestimmte Aktion auslösen (man erinnere sich an das Eingangsbeispiel – dort waren in der Dialog-Box zwei derartige Knöpfe positioniert).

Der Push-Button ist unter dem Symbol  abgelegt:

Push-Button („Schaltfläche”)



Diesen klickt man an und zieht ihn mit links gedrückter Maustaste auf das freie Feld des GUI:

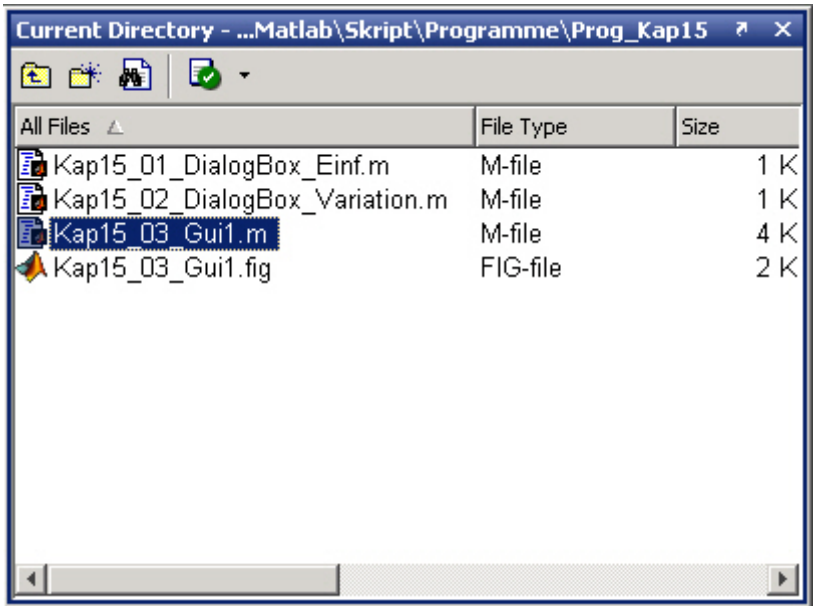


Damit lässt sich natürlich (noch) nicht (allzu) viel anfangen.

Den GUI sollte man abspeichern. Dies ergibt eine sogenannte Layout-Datei, die durch das Kürzel „fig“ gekennzeichnet ist.

Wir wollen die Datei „Kap15_03_Gui1“ nennen. Es wird also abgespeichert die Layout-Datei „`Kap15_03_Gui1.fig`“.

Dem aufmerksamen Verfolger der Aktion wird an dieser Stelle wahrscheinlich nicht entgangen sein, dass noch eine weitere Datei erzeugt wurde, wie auch aus der Auflistung im aktuellen Verzeichnis von MATLAB angezeigt wird:



Neben der Layout-Datei `Kap15_03_Gui1.fig` findet sich auch die (fast) gleichnamige Datei `Kap15_03_Gui1.m`, eine (fast) normale Datei zur Aufnahme eines MATLAB-Programms. Sie wird auch angezeigt und erschreckt mit folgendem Inhalt:

```
function varargout = Kap15_03_Gui1(varargin)
% KAP15_03_GUI1 M-file for Kap15_03_Gui1.fig
%      KAP15_03_GUI1, by itself, creates a new KAP15_03_GUI1 or raises the
%      existing singleton*.
%
%      H = KAP15_03_GUI1 returns the handle to a new KAP15_03_GUI1 or the %
%      handle to the existing singleton*.
%
%      KAP15_03_GUI1('CALLBACK',hObject,eventData,handles,...) calls the %
%      local function named CALLBACK in KAP15_03_GUI1.M with the given
%      input arguments.
%
%      KAP15_03_GUI1('Property','Value',...) creates a new KAP15_03_GUI1 or
%      raises the existing singleton*.  Starting from the left, property
%      value pairs are applied to the GUI before
%      Kap15_03_Gui1_OpeningFunction gets called.  An unrecognized property
%      name or invalid value makes property application stop.  All inputs
%      are passed to Kap15_03_Gui1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_03_Gui1

% Last Modified by GUIDE v2.5 29-May-2007 16:53:43

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_03_Gui1_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_03_Gui1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
```

```
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_03_Gui1 is made visible.
function Kap15_03_Gui1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_03_Gui1 (see VARARGIN)

% Choose default command line output for Kap15_03_Gui1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_03_Gui1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_03_Gui1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Wer bis hierher vorgedrungen ist, hat automatisch das Ende der Datei Kap15_03_Gui1.m erreicht. Hier am Schluß steht der Teil, der momentan von Interesse ist – die letzte Funktion

```
function pushbutton1_Callback(hObject, eventdata, handles)
```

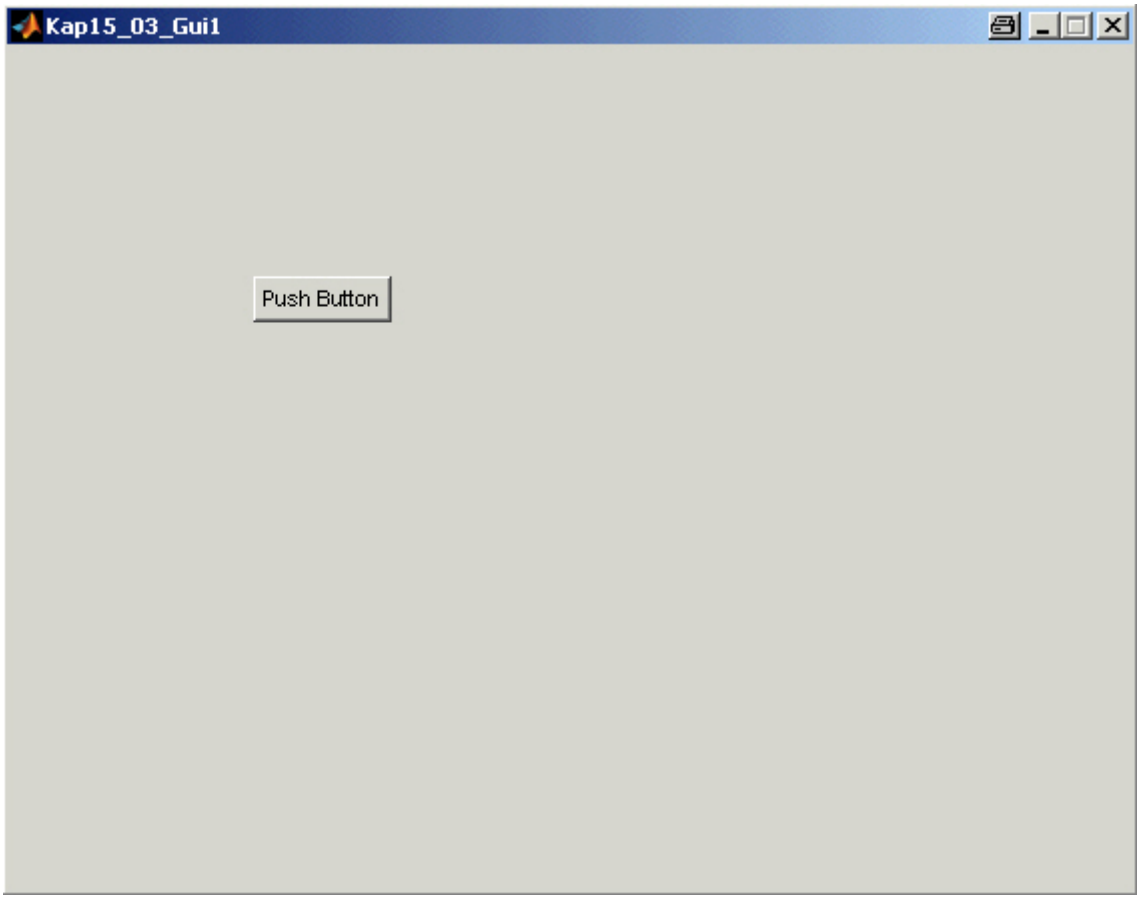
Dies ist die Rückruf-Funktion oder Callback, die aufgerufen wird, wenn der Anwender später den mühsamst installierten Push-Button per Mausklick drückt.

Wir können den GUI sogar aufrufen.

Dazu tippen wir im Kommandofenster ein

```
>> Kap15_03_Gui1
```

Es dauert einen kleinen Moment, aber dann erscheint folgendes Fenster:



Sieht nicht sonderlich überzeugend aus, aber wir haben in Graphik, Übersicht, Beschriftung usw. noch nicht sonderlich viel investiert. Genaugenommen haben wir noch gar nichts investiert, denn das Anklicken des „Push Button“ erzeugt offenbar keine Reaktion.

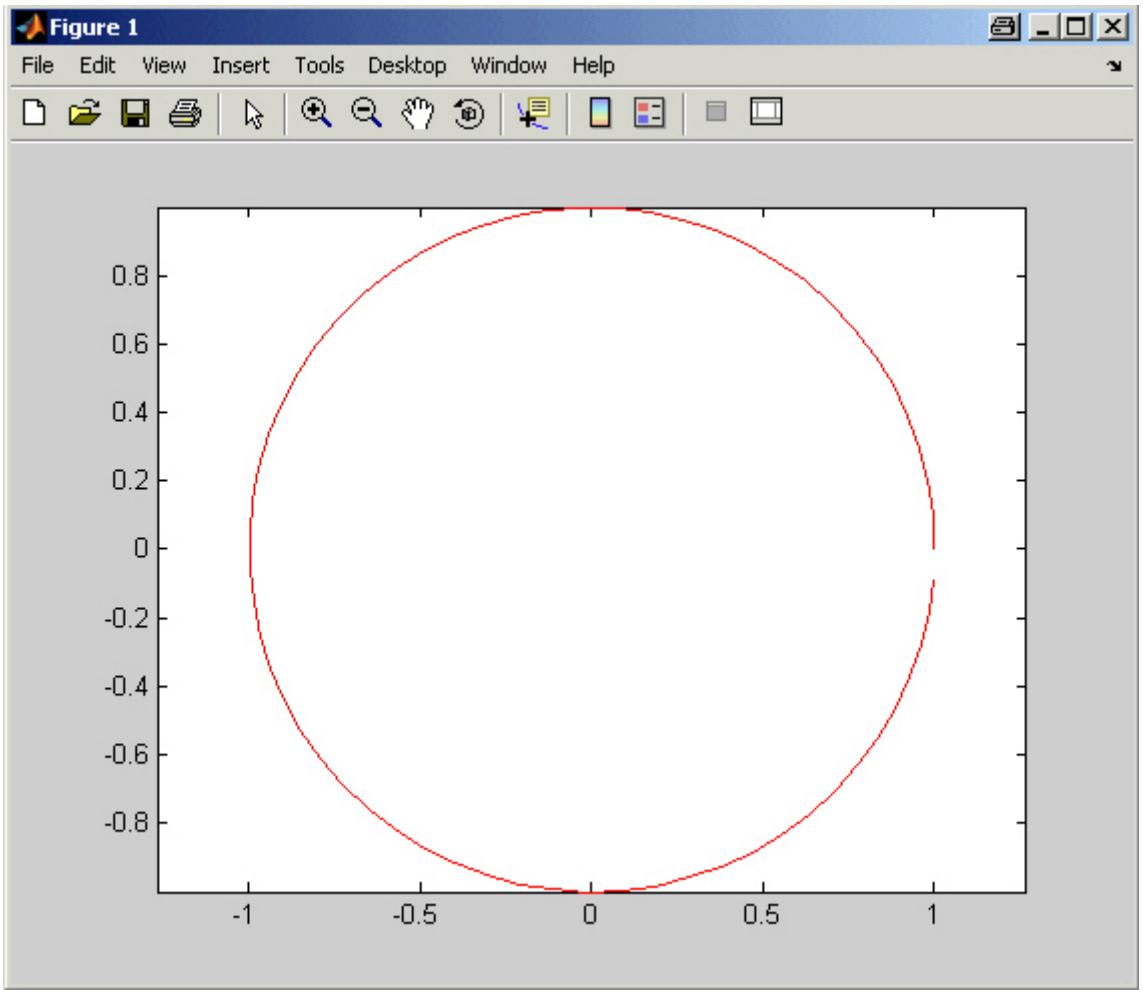
Dies wiederum ist nicht verwunderlich:

Zwar existiert die Rückruf-Funktion `pushbutton1_Callback`, aber die ist noch leer.

Also werden wir eine Reaktion dadurch erhalten, indem wir diese Funktion wie folgt erweitern:

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
figure;
t = 0:0.1:2*pi;
x = cos(t); y = sin(t);
plot(x, y, 'r');
axis equal;
```

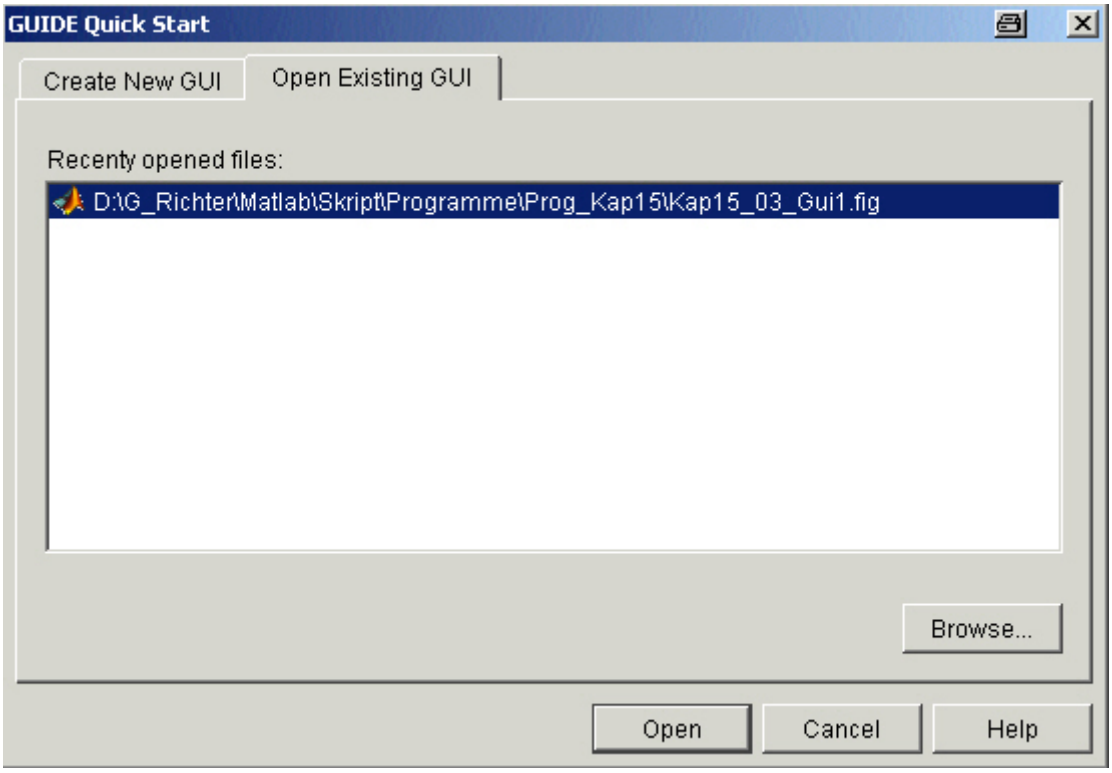
Dann bringt ein Anklicken der Schaltfläche „Push Button“ auch tatsächlich einen schönen roten Kreis auf den Bildschirm:



Modifikation des GUI:

Zwar war die Aktion recht erfolgreich, doch sieht der GUI noch etwas arg schäbig aus. Einige Änderungen werden nicht schaden. Dazu muß die Layout-Datei des GUI erneut aufgerufen werden.

Eintippen ins Kommandofenster „guide“ bringt erneut das Fenster „GUIDE Quick Start“ auf den Bildschirm mit dem Tabreiter „Create New GUI“. Diesmal wählen wir jedoch den anderen „Open Existing GUI“:



Die Schaltfläche „Open“ (auch dieser ist ein „Push Button“) öffnet wieder die Layout-Datei `Kap15_03_Gui.fig`. Diese können wir modifizieren.

Beispielsweise ist der Name „Push Button“ für die Schaltfläche nicht sonderlich aussagekräftig. Mit dem Cursor geht man auf den Knopf „Push Button“, bis der Zeiger sich in ein Kreuz verwandelt. Ein Klick der rechten Maustaste öffnet ein Kontext-Menü. Man klicke den Eintrag „Property Inspector“ an. Es öffnet sich ein weiteres (recht umfangreiches) Fenster mit den Eigenschaften, die bzgl. der Schaltfläche gesetzt werden können.

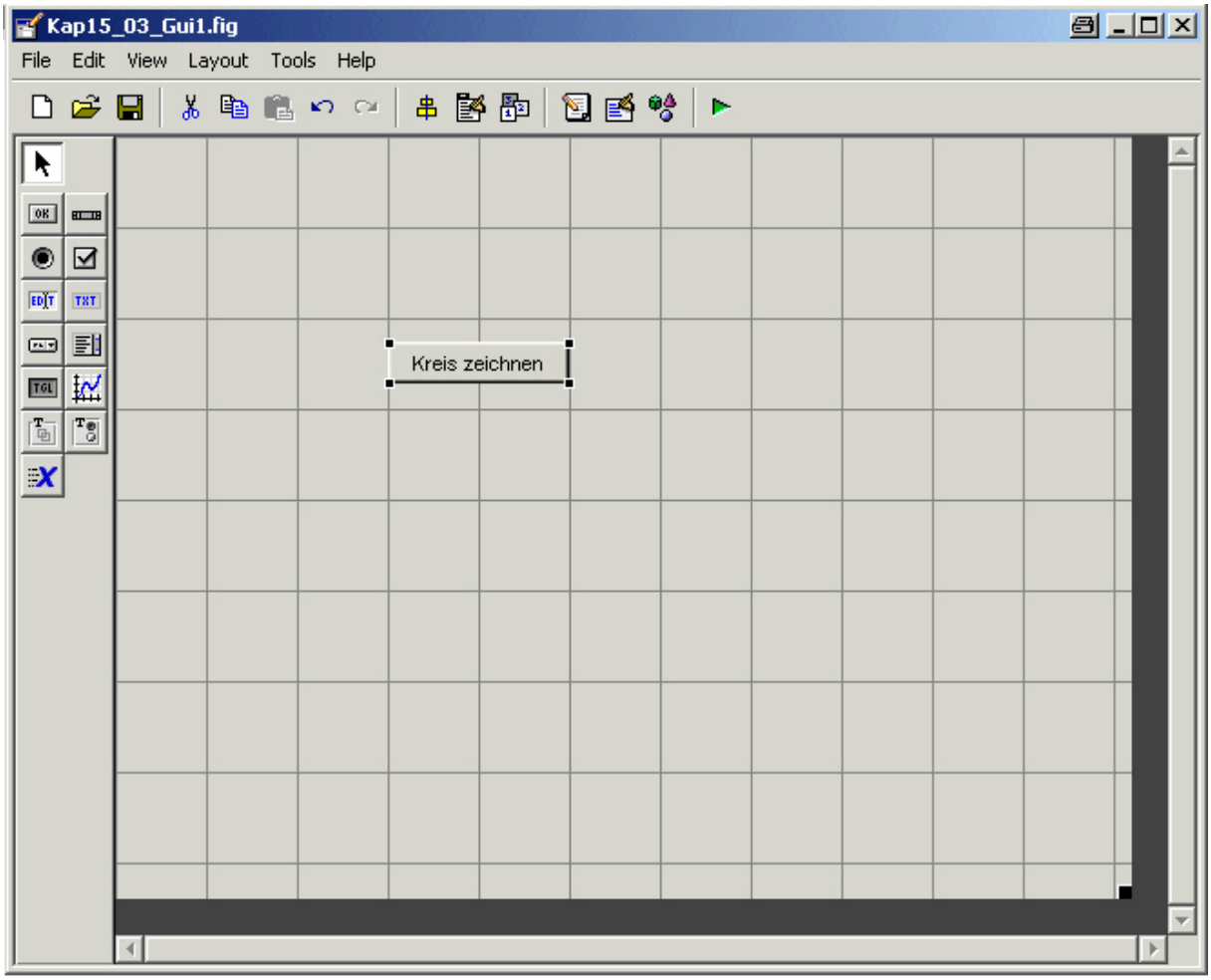
Davon setzen wir einen Wert:

„String“: Kreis zeichnen

„Position.width“: 20

Die Schaltfläche erhält damit den Namen „Kreis zeichnen“ und – da dieser Eintrag doch etwas lang ist, die über die aktuelle Länge des Knopfes hinwegreicht – einen etwas längere Abmessung:

Damit hat der GUI jetzt folgendes Aussehen:

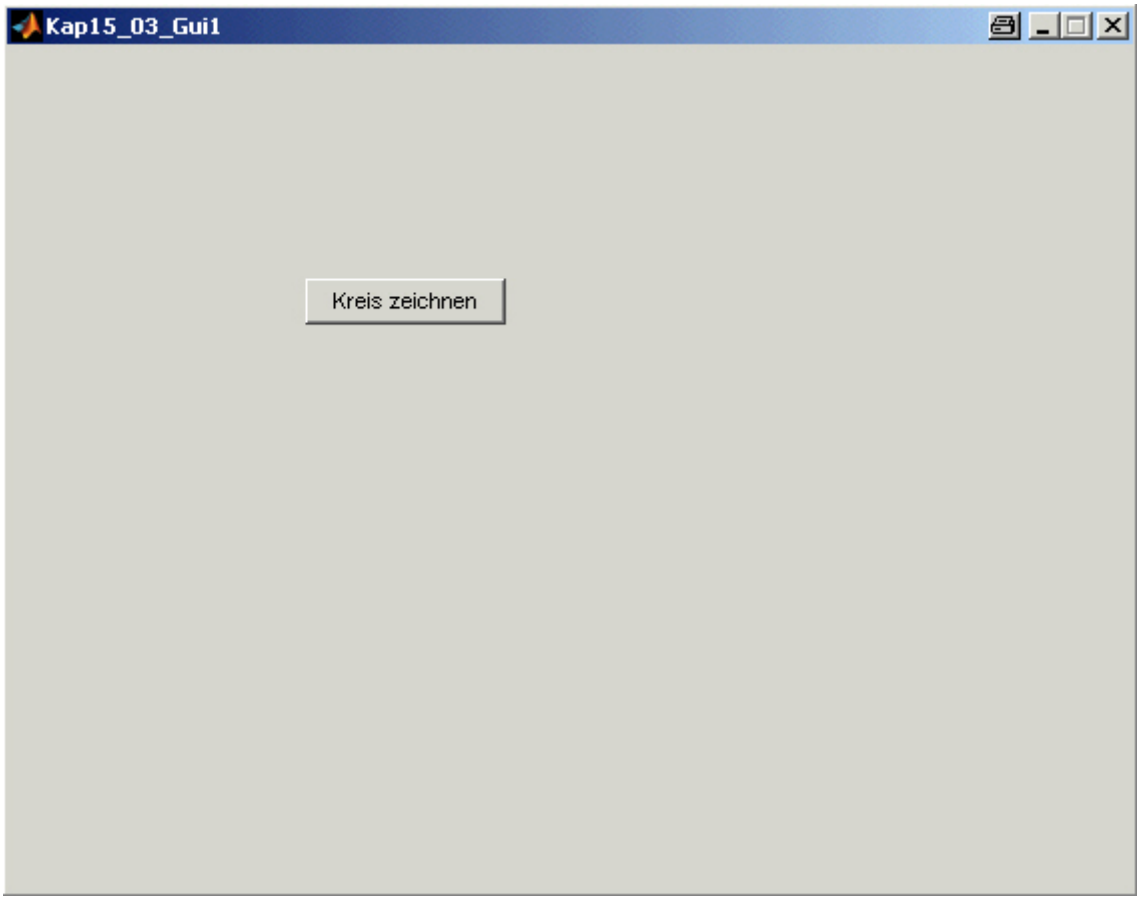


Die Änderungen müssen natürlich abgespeichert werden. Da fragt das System schon automatisch nach.

Wird dann wieder über das Kommandofenster aufgerufen:

```
Kap15_03_Gui1
```

so wird die modifizierte Fassung auf den Bildschirm gebracht:

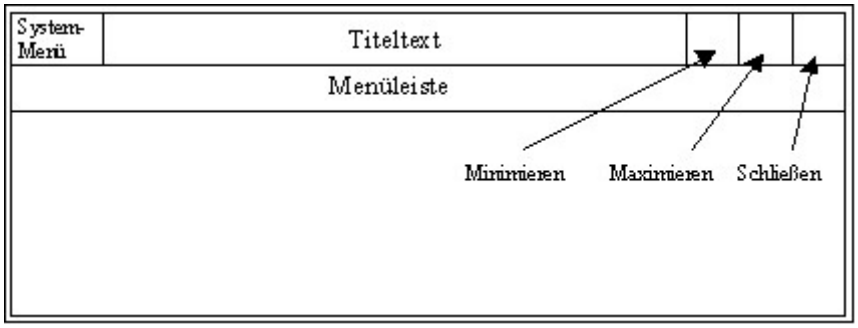


Die Abmessungen des GUI's lassen sich ebenfalls modifizieren, indem man (wieder in GUIDE) mit dem Cursor an die rechte untere Ecke des GUI geht – diese ist mit einem schwarzen Punkt markiert – und dann bei gedrückter linker Maustaste den GUI zusammenschiebt oder auseinanderzieht – doch wir lassen es hier erst einmal so, wie es ist.

Fenster als GUI

Eine graphische Benutzeroberfläche – also ein GUI – stellt sich dem Anwender als ein Fenster mit verschiedenen Elementen dar. Hinter dieser Darstellung liegt als Datentyp ein Objekt zugrunde, ein sog. UI-Objekt („user-interface"). Dies verwaltet als Elemente die Daten und Methoden, die mit dem Fenster in Verbindung stehen. Das meiste davon läuft für den Anwender unsichtbar im Hintergrund ab, doch bestimmte Teile sind von ihm zu ergänzen bzw. auszufüllen.

Ein solches Fenster ist in etwa wie folgt aufgebaut:



Sämtliche Aktionen und Reaktionen spielen sich also jetzt über das Fenster ab (oder für anglophile „windows“):

Es werden mit der Maus Elemente angeklickt oder Inhalte in Eingabefelder geschrieben; Ausgaben erfolgen in Ausgabefeldern, oder es werden Meldungsfenster geöffnet (die Dialog-Box lernten wir ja bereits kennen).

Dies ist der Teil, den der Programmierer implementieren muß. Wie allerdings schon am ersten GUI gezeigt, läuft sehr viel automatisch ab, so dass auf weite Strecken „nur“ noch ausgefüllt werden muß. Allerdings setzt das Ausfüllen die Kenntnis dessen aus, was denn ausgefüllt werden soll. Im wesentlichen wird das Verhalten eines GUI’s neben dem Property Inspector in GUIDE von dem dazugehörigen Programm, abgelegt in einem m-File gleichen Namens, gesteuert. Ein solches Programm (genaugenommen sogar zwei) haben wir bereits erzeugt bei Erzeugung eines GUI’s mit GUIDE: in dem Moment, wo „Speichern“ des in GUIDE erstellten neuen GUI’s gedrückt wurde, wurde nicht nur ein Layout-File mit Kürzel „fig“, sondern auch ein m-File (mit Kürzel „m“) angelegt.

Letzteren sehen wir uns erneut an:

Der GUI-m-File

Bei Erstellung eines GUI’s in GUIDE wird automatisch mit dem Layout-File ein m-File angelegt (Standardbenennung MATLAB: „`gui1.fig`“, `gui1.m`“). Im letzten Beispiel hieß er „`Kap15_03_Gui1.m`“.

Sehen wir uns diese Datei noch einmal an (einige Kommentare wurden der Übersicht wegen weggelassen):

```
function varargout = Kap15_03_Gui1(varargin)
% KAP15_03_GUI1 M-file for Kap15_03_Gui1.fig

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_03_Gui1_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_03_Gui1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_03_Gui1 is made visible.
function Kap15_03_Gui1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_03_Gui1 (see VARARGIN)

% Choose default command line output for Kap15_03_Gui1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_03_Gui1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_03_Gui1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
figure;
t = 0:0.1:2*pi;
x = cos(t); y = sin(t);
plot(x, y, 'r');
axis equal;
```

Dies ist im Wesentlichen der Inhalt der Datei, die die Funktionalität des ersten GUI's regelt, der nichts weiter enthält als einen einfachen simplen Knopf.

Die Datei enthält vier Funktionen (der Kundige wird sie bereits an dem Schlüsselwort `function` erkannt haben):

`Kap15_03_Gui1`	Startfunktion, mit der der Anwender das Fenster aufruft
`Kap15_03_Gui1_OpeningFcn`	automatisch aufgerufen direkt vor Fensteraufbau
`Kap15_03_Gui1_OutputFcn`	aufgerufen vor der Abfrage des Rückgabewertes
`pushbutton1_Callback`	aufgerufen, wenn der Knopf im GUI gedrückt wird

Startfunktion `Kap15_03_Gui1`:

Nicht umsonst steht am Anfang „`% Begin initialization code - DO NOT EDIT`" – und als brave Kinder halten wir uns daran.
Der Prozess ist ohnehin automatisiert.
Doch was geschieht hier?
Es wird eine Struktur angelegt mit den Öffnungs- und Schließfunktionen als Elementen. Das Layout merkt sich auf diese Weise relevante Daten – insbesondere, welche Funktionen beim Öffnen und beim Schließen aufgerufen werden müssen.
In unserem Beispiel ist hier `Kap15_03_Gui1_OpeningFcn` und `Kap15_03_Gui1_OutputFcn` eingetragen.
Mit diesen Daten wird dann die Hauptfunktion `gui_mainfcn` aufgerufen (dazu gibt es zwei Versionen – einmal ohne Übergabeparameter und einmal mit – braucht uns aber nicht weiter zu kümmern.
Diese Funktion nehmen wir also nur zur Kenntnis. An die Angabe „`DO NOT EDIT`" halten wir uns.

Öffnungsfunktion `Kap15_03_Gui1_OpeningFcn`:

Diese wird kurz vor Öffnen des Fensters aufgerufen, und zwar von `gui_mainfcn`. Sie stellt nämlich die Daten für das Fenster bereit. `gui_mainfcn` übergibt dabei bei Aufruf von `Kap15_03_Gui_OpeningFcn` folgende Parameter:

`hObject:`	Handle auf das Objekt figure, das das Hauptfenster repräsentiert
`eventdata:`	zur Zeit nicht verwendet

`handles:`	Struktur mit den Daten des Fensters und weiteren Daten des Anwenders
`varargin`	(„variablen argument input“) beim Aufruf von `Kap15_03_Gui1` übergebene weitere Argumente.

Was tatsächlich vom Anwender gefüllt werden kann (und sollte) ist die Struktur `handles`; diese können dann in den Rückruf-Funktionen der Fenster-Elemente verwendet zu werden (kommt noch im nächsten Beispiel, wie das geht).
Die Daten in `handles` werden mit der Funktion `guidata` zum aktuellen Fenster `figure` (Objekt, das das Hauptfenster repräsentiert) gespeichert:

```
guidata(hObject, handles);
```

Dann folgt eine auskommentierte Anweisung:

```
% uiwait(handles.figure1);
```

Die wird später noch verwendet werden – doch im Moment lassen wir’s so, wie es ist.

Rückgabefunktion `Kap15_03_Gui1_OutputFcn:`

Die Aufgabe der Funktion `Kap15_03_Gui1_OutputFcn` besteht hauptsächlich darin, die Werte bereitzustellen, die `Kap15_03_Gui1` zurückgeben soll – welche das sind, betrachten wir später. Die Rückgabe erfolgt über die Sammelvariable `varargout`; in dieser können die unterschiedlichsten Variablen abgelegt werden. Standardmäßig ist dort das Output-Element der Stuktur handle eingetragen:

```
varargout{1} = handles.output;
```

Dieser Wert `varargout` ist der, der – siehe oben im Beispiel – ist der Rückgabewert von `Kap15_03_Gui1`:

```
function varargout = Kap15_03_Gui1(varargin)
```

Dieser wird belegt, bevor `Kap15_03_Gui1` diesen Wert (an wen auch immer) zurückgibt.
Es lässt sich auch folgendermaßen sehen:
Normalerweise startet man das GUI durch Aufruf der Startfunktion, also in unserem Beispiel durch Aufruf

```
>> Kap15_03_Gui1
```

Statt dieser Eingabe machen wir jetzt

```
>> h = Kap15_03_Gui1
```

Abgesehen, dass sich damit das GUI-Fenster mit dem einzigen Knopf „Kreis zeichnen“ öffnet, antwortet das System mit:

```
h =

  157.0012

>>
```

157.0012 ist die Bezugsnummer – eben der Handle des Objekts, das durch Aufruf von `Kap15_03_Gui1` erzeugt wurde.

Wir modifizieren ein wenig die `OutputFcn`:

```
function varargout = Kap15_03_Gui1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
%varargout{1} = handles.output;
varargout{1} = 'Mein GUI';
```

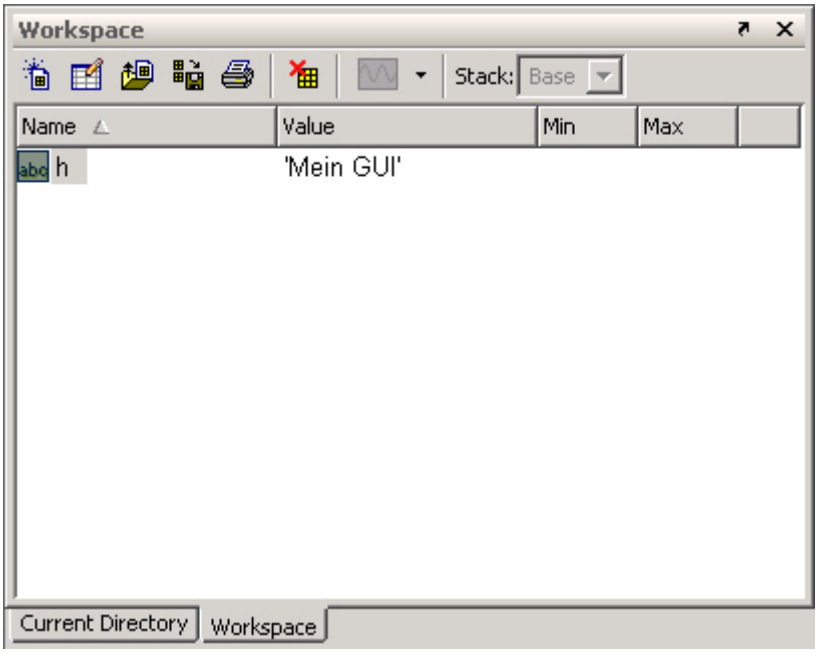
Rufen wir erneut im Kommandofenster auf

```
>> h = Kap15_03_Gui1
```

erscheint folgende Abfolge

```
>> h = Kap15_03_Gui1

h =

Mein GUI

>>
```

Auch im Fenster, das den Arbeitsspeicher anzeigt, ist jetzt zu sehen:



Dieser Rückgabewert wird sofort nach Aufruf von `Kap15_03_Gui1` im Kommandofenster gesetzt. Damit man ihn noch während der Arbeit mit dem GUI ändern kann (also nicht nur in der Funktion `Kap15_03_Gui1_OutputFcn`, muß in der Aufruf-Funktion `Kap15_03_OpeningFcn` der Aufruf `uiwait` gesetzt werden – kommt noch.

Die Rückruf-Funktion `pushbutton1_Callback`:

Wie bereits erwähnt, wird diese Funktion aufgerufen, wenn der Knopf im GUI mittels Maus gedrückt wird.

Auch diese Funktion bekommt ähnlich wie die Öffnungsfunktion `Kap15_03_Gui1_OpeningFcn` Parameter übergeben:

`hObject`: Handle der Elemente, hier der Knopf

`eventdata`: zur Zeit nicht verwendet

`handles`: Struktur mit Daten des Fensters und weiteren Daten des Anwenders.

In unserem Beispiel wurde zusammenhanglos ein Kreis gezeichnet, ohne auf bereits vorhandene Daten zurückzugreifen. Es ist jedoch möglich (um nicht zu sagen, häufig notwendig), auf bereits bestehende Daten zuzugreifen und diese in die Aktion mit einzubeziehen. Dies geschieht über die Daten in `handles`. Werden diese geändert, muß immer die Funktion `guidata` aufgerufen werden, um die Änderungen von `handles` im Object `hObject` zu speichern:

```
guidata(hObject, handles);
```

Dies sind die vier Funktionen, die bei Erzeugung eines GUI-Objekts in GUIDE automatisch mit angelegt werden.

Die Reihenfolge ihres Auftretens ist die folgende:

1. im Kommandofenster wird der Befehl eingegeben:
   ```
   >> h = Kap15_03_Gui1
   ```
2. Es wird aufgerufen
   ```
   varargout = Kap15_03_Gui1(varargin)
   ```
   Da wir nur `h = Kap15_03_Gui1` aufgerufen haben, gibt es keine Eingabeparameter, d.h. `varargin` ist leer, (was durch `nargin` abgefragt werden kann), und es gibt einen Rückgabeparameter (nämlich h; was über `nargout` abgefragt werden kann).
   In der Funktion wird gesetzt wird (u.a.)
   ```
   gui_State = struct('gui_Name',       mfilename, ...
   ```

```
                    'gui_Singleton',  gui_Singleton, ...
                    'gui_OpeningFcn', @Kap15_03_Gui1_OpeningFcn, ...
                    'gui_OutputFcn',  @Kap15_03_Gui1_OutputFcn, ...
```

Es ist hier die Anzahl der Rückgabewerte 1 (nämlich h wegen des Aufrufs `h = Kap15_03_Gui1`); also ist `nargout` = 1.

Daher ist im Programmteil

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

der `if`-Zweig erfüllt.

Es wird also das Hauptprogramm `gui_mainfcn` mit dem hier belegten `gui_State` übergeben (`varargin` ist leer). `varargout` wird erst belegt, wenn `gui_mainfcn` abgearbeitet ist – das ist aber noch nicht der Fall.

3. Also wird aufgerufen `gui_mainfcn` mit der Struktur gui_State.
   a. `gui_mainfcn` wiederum ruft auf
      `Kap15_03_Gui1_OpeningFcn(hObject, eventdata, handles, varargin)`
      Diese Funktion stellt Daten für das Fenster bereit
      erst wenn sie abgearbeitet ist, wird der GUI auf den Bildschirm gebracht.
   b. Sie ist abgearbeitet; Rücksprung nach `gui_mainfcn`.
   c. Der GUI erscheint auf dem Bildschirm (aufgerufen von `gui_mainfcn`)
   d. `gui_mainfcn` ruft auf:
      `varargout = Kap15_03_Gui1_OutputFcn(hObject, eventdata,handles)`
      In dieser Funktion wird `varargout{1}` auf „Mein GUI“ gesetzt.
      Sie wird beendet und gibt `varargout` an die aufrufende Funktion `gui_mainfcn` zurück.
   e. `gui_mainfcn` wird beendet und gibt `varargout` an das die aufrufende Funktion `Kap15_03_Gui1(varargin)`.
4. `Kap15_03_Gui1(varargin)` wird beendet und gibt `varargout` an das aufrufende Kommandofenster zurück. Im Kommandofenster steht jetzt

```
h =

Mein GUI


>>
```

und der GUI wartet brav auf weitere Eingaben.

In einer graphischen Übersicht lässt sich die Abfolge wie folgt darstellen:

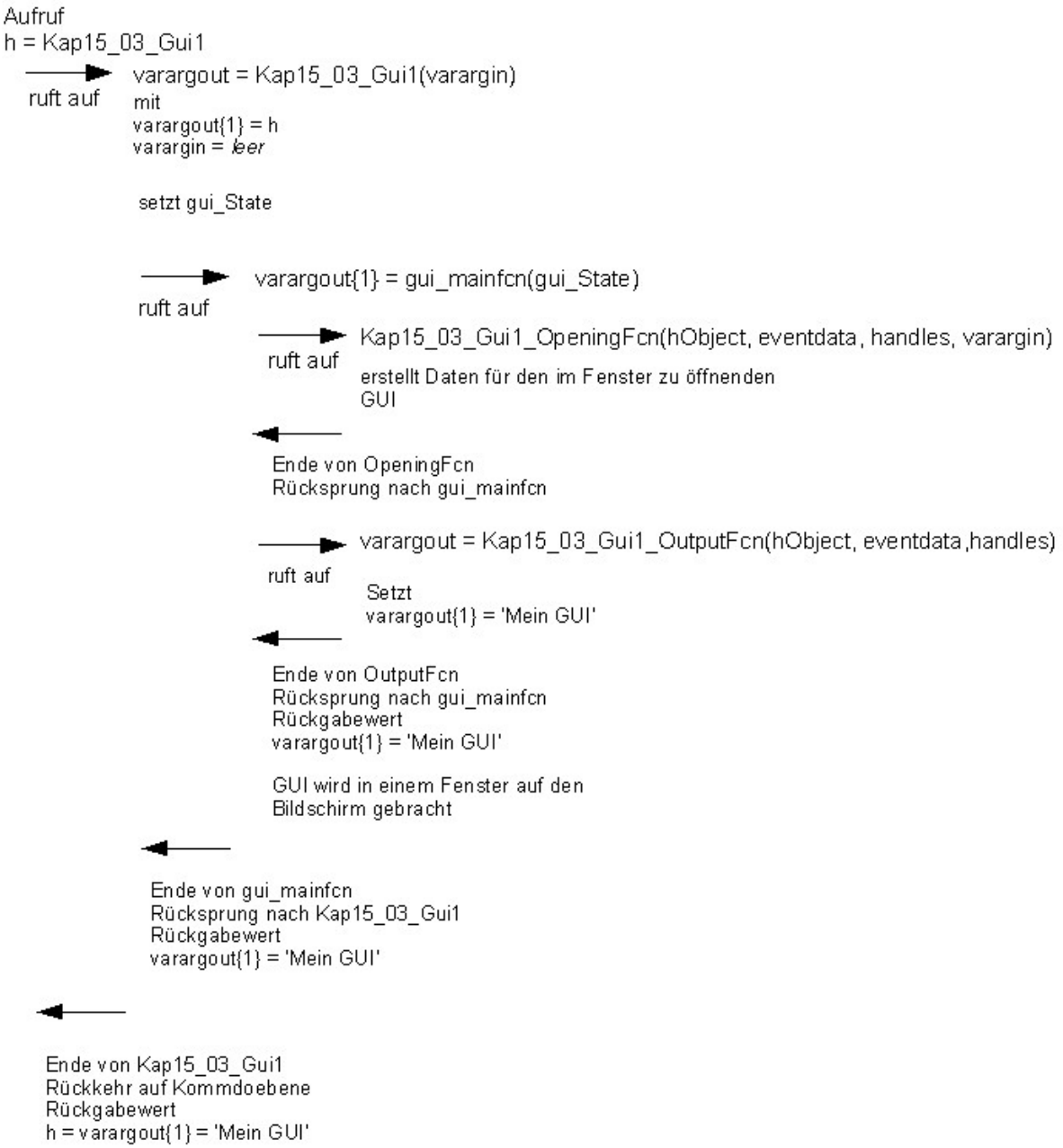


Abbildung 1

Befehlsfolge der Funktionen zur Darstellung des GUI-Objekts nach Aufruf

h = Kap15_03_Gui1 im Kommandofenster

Diese detaillierte Aufstellung der Abfolge der Aufrufe der Funktionen wird uns später das Verständnis erleichtern, wenn es um die Rückgabe von GUI-Werten geht.

Wird nun über Mausklick eine Aktion auf dem GUI ausgelöst, ist die Abfolge des Aufrufs der Funktionen die folgende

1. Im GUI wird per Mausklick der Knopf „Kreis zeichnen“ gedrückt.
2. Aufruf von `varargout = Kap15_03_Gui1(varargin)` mit vier Eingabeparametern, aber ohne Rückgabeparameter. Der erste Eingabeparameter lautet:

   `varargin{1} = pushbutton1_Callback`

   Somit ist die `if`-Bedingung

   ```
   if nargin && ischar(varargin{1})
       gui_State.gui_Callback = str2func(varargin{1});
   end
   ```

   erfüllt. Also wird in der Struktur `gui_State` die Rückruf-Funktion auf den Wert von `varargin{1}`, also auf „`pushbutton1_Callback`“ gesetzt (bei Öffnen des GUI’s war dieser Wert leer geblieben).
3. Damit wird `gui_mainfcn` mit Übergabegabeparameter `guiState` (also insbesondere der Rückruf-Funktion „`pushbutton1_Callback`“), aber ohne Rückgabeparameter aufgerufen.
   a. `gui_mainfcn` ruft nun

      `pushbutton1_Callback(hObject, eventdata, handles)`

      auf.
      i. Diese Funktion öffnet ein neues Fenster, um den Kreis zu zeichnen.
      ii. Damit ist `pushbutton1_Callback(hObject, eventdata, handles)` abgearbeitet
      iii. Rücksprung nach `gui_mainfcn`
   b. Rücksprung nach `Kap15_03_Gui1(varargin)`.
4. `Kap15_03_Gui1(varargin)` ist abgearbeitet und damit die Aktion des Knopf-Drückens abgeschlossen.

Die Abfolge läßt sich graphisch in etwa so darstellen:

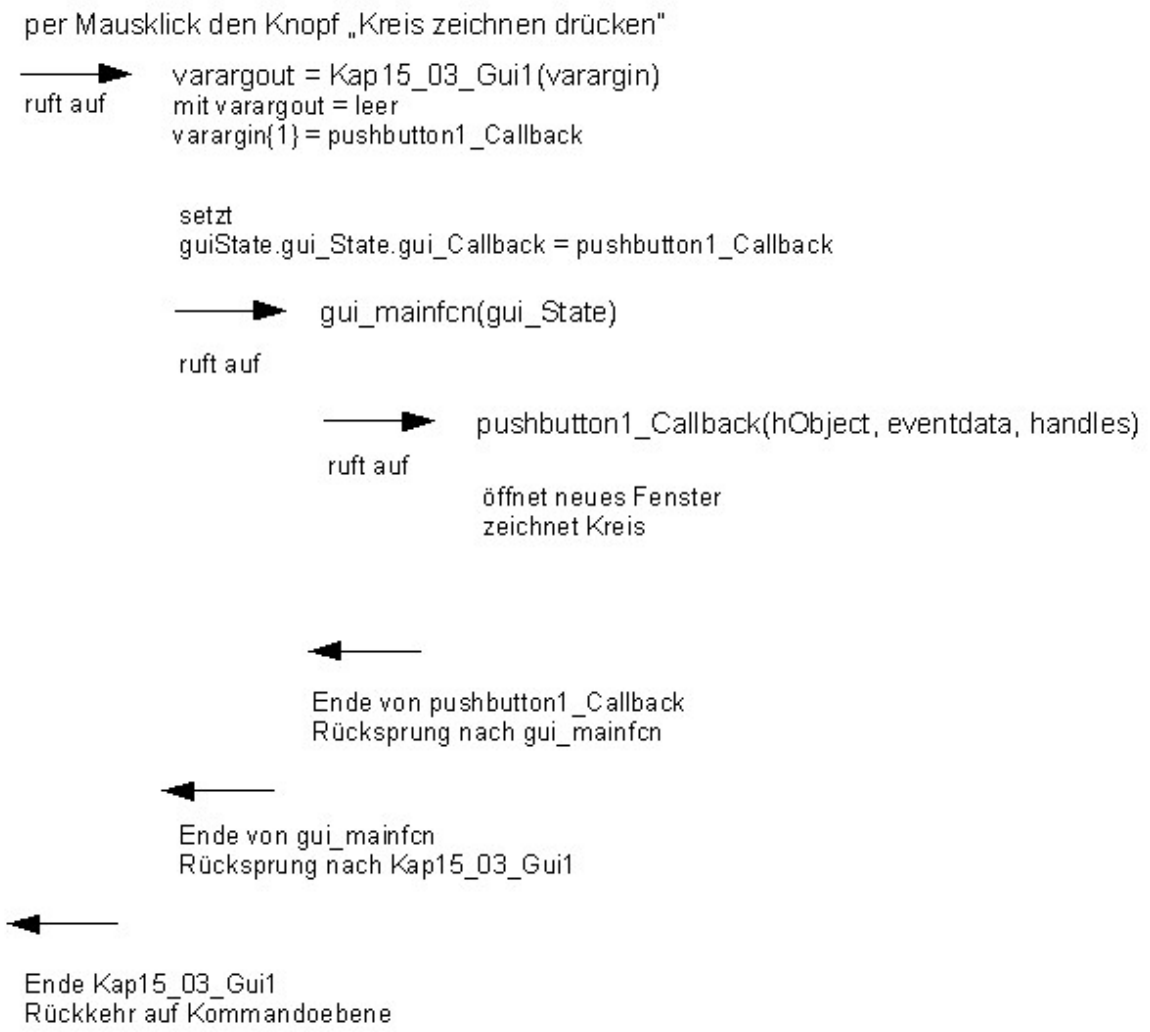


Abbildung 2 Abfolge der Funktionen nach Betätigen des Knopfes "Kreis zeichnen" per Mausklick

Wird der GUI geschlossen, wird (in diesem Beispiel) keine der hier aufgeführten Funktionen aufgerufen.

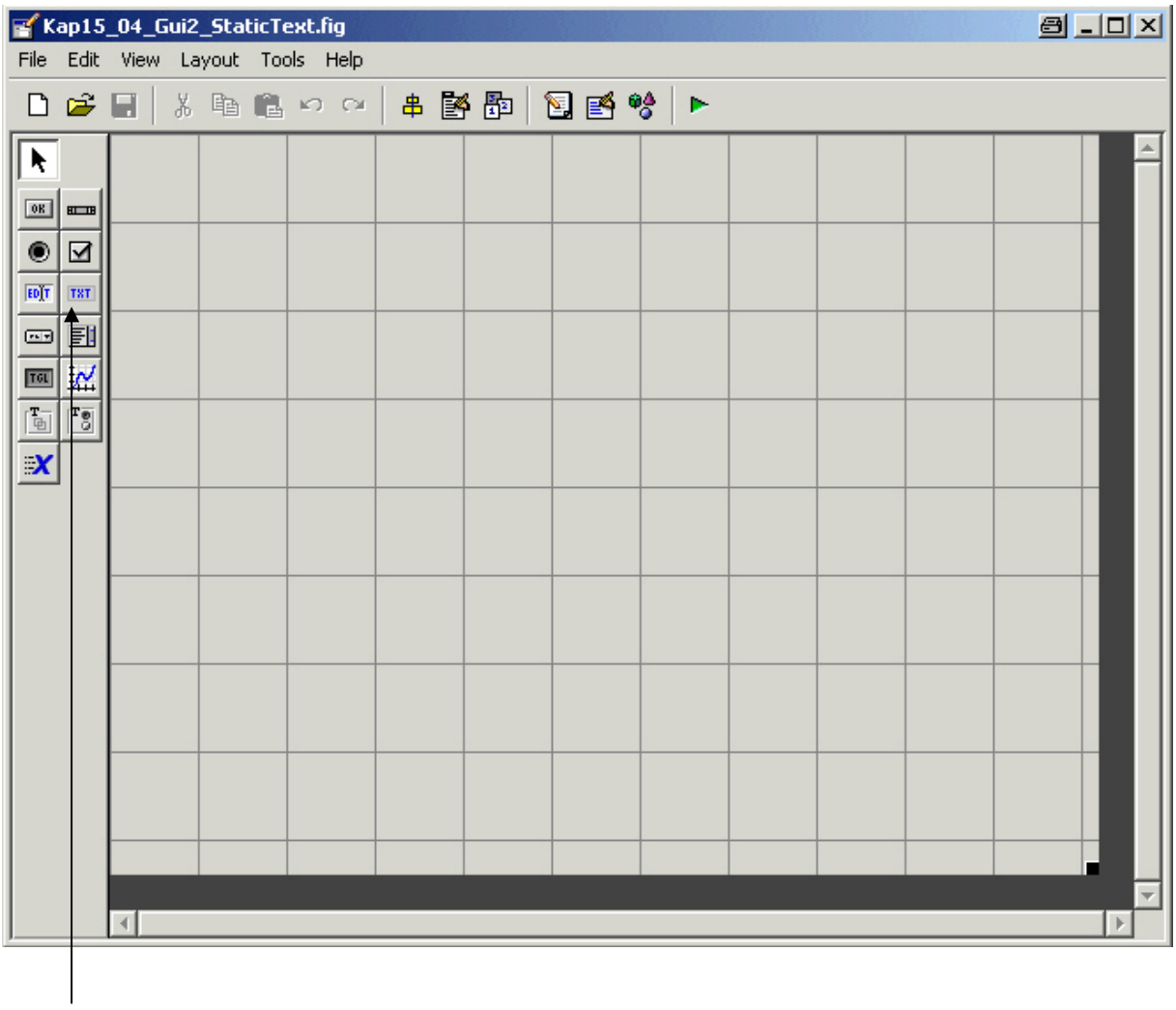
Bei komplexeren Elementen in einem GUI wird natürlich die gegenseitigen Aufrufe der Funktionen ebenfalls umfangreicher werden.
Dies werden wir jetzt Schritt für Schritt untersuchen.

Statischer Text in einem GUI

Als nächstes Beispiel betrachten wir die Möglichkeiten, einen Text auf einen GUI zu schreiben. Dieser Text befindet sich bereits in Form einer Beschriftung auf dem GUI, wenn dieser über das Kommandofenster aufgerufen wird und kann dann auch nicht mehr über den

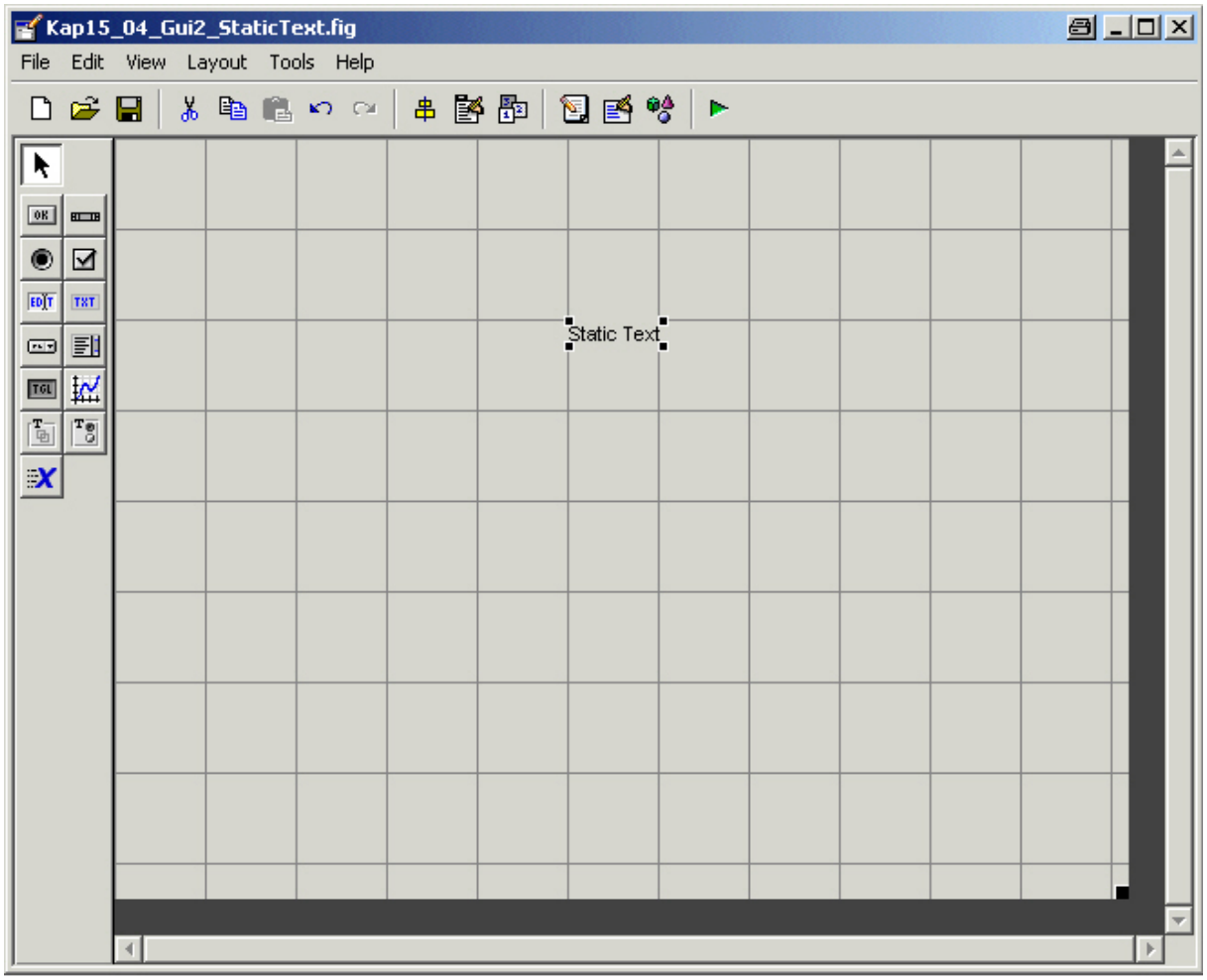
GUI selber geändert werden. Ein solcher Text heißt „static text“ und wird mit der gleichnamigen Schaltfläche in der GUIDE erzeugt.

Wieder einmal rufen wir die GUIDE auf und lassen diese ein leeres Layout für einen neuen GUI anlegen, den wir unter „`Kap15_04_Gui2_StaticText.fig`“ abspeichern.



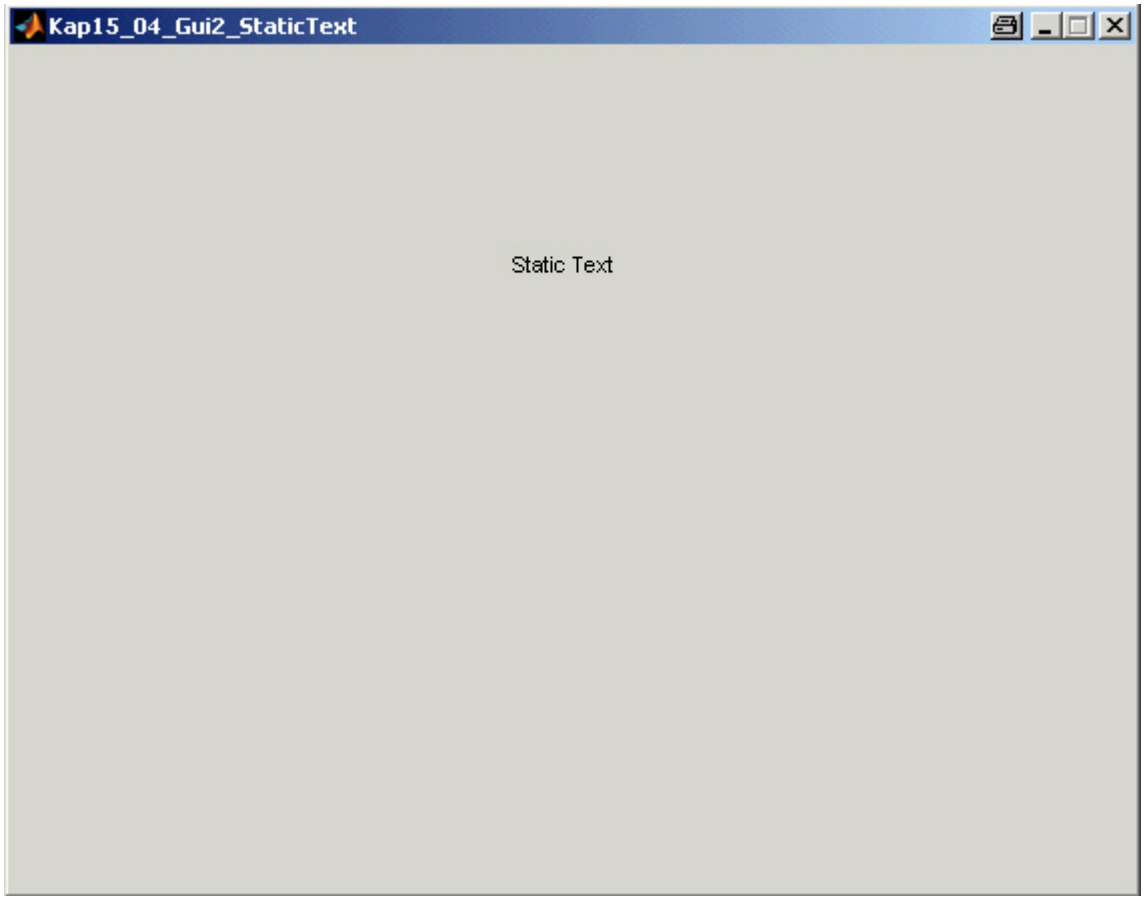
Static Text

In der zweiten Spalte befindet sich in der Auswahlreihe der GUI-Elemente dasjenige für „Static Text“. Diesen mit der linken Maustaste anklicken (rechte geht auch) und bei gedrückter Maustaste in das Layout-Feld ziehend erhält man dann ein Feld mit dem vielsagenden Inhalt „Static Text“



Betrachtet man sich den dazugehörigen m-File „`Kap15_04_Gui2_StaticText.m`“, wird man nicht sehr viel Erhellendes feststellen – er ist bis auf die Rückruf-Funktion nahezu naturidentisch mit dem von „`Kap15_03_Gui1.m`“ vor der Erweiterung seitens des Anwenders. Daß es hier keine Rückruf-Funktion gibt, ist in der Tat nicht abwegig: ein Ausgabebefehl reagiert nicht auf Maus-Klicks, braucht auch keine Funktion, die darauf reagiert.

Ruft man also „`Kap15_03_Gui2_StaticText`“ im Kommandofenster auf, erscheint das langweilige GUI der folgenden Form:

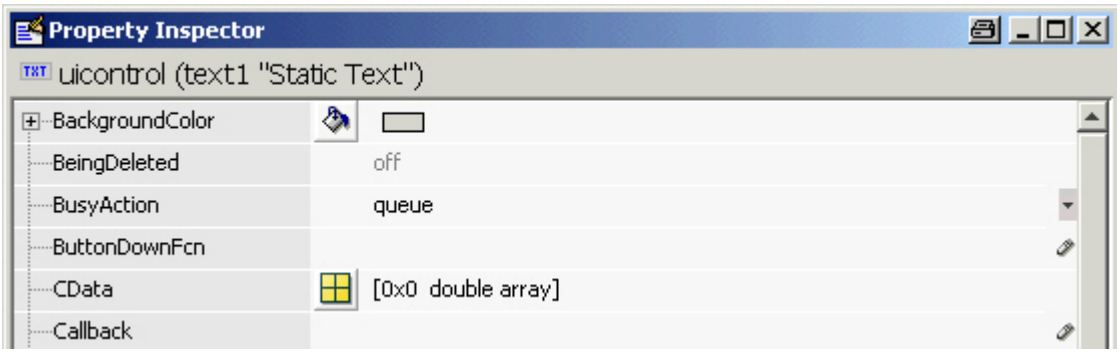


Abgesehen von der Positionierung (die auch geändert werden kann), ist wahrscheinlich die Vorbelegung „`Static Text`“ nicht so unbedingt die gewünschte Textausgabe. Um den Text anzupassen, gibt es zwei Möglichkeiten:

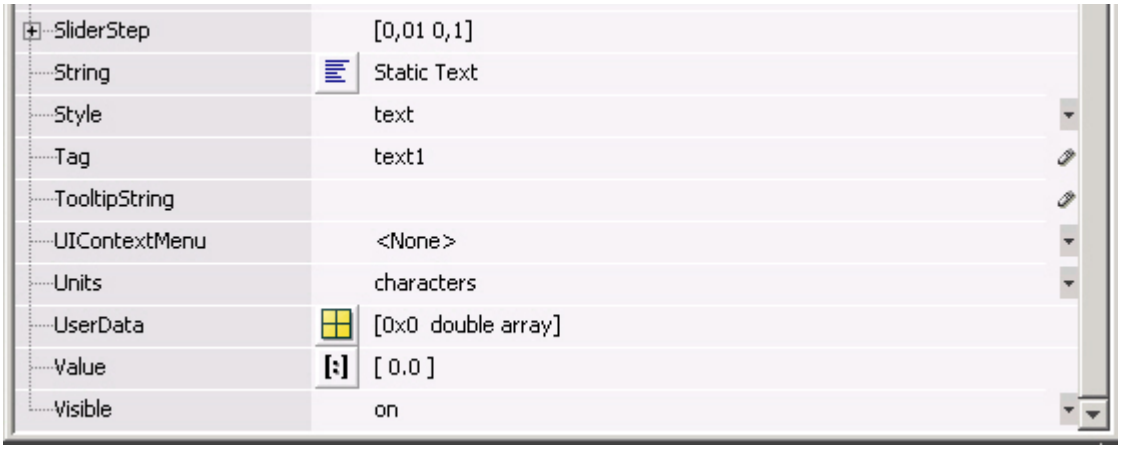
- Im Layoutfenster kann ein beliebiger Text fest eingetragen werden
- Das Textfeld wird zur Laufzeit mit einem Text gefüllt

Für die erste Möglichkeit ruft man im Layout-Fenster von GUIDE den „Property-Inspector“ auf – dies erreicht man durch Anklicken des Feldes, bis sich der Cursor in ein Kreuz mit Pfeilenden verwandelt und man dann die rechte Maustaste klickt.
Es öffnet sich ein Fenster mit einem Kontext-Menü.
In diesem wählt man aus „Property Inspector“

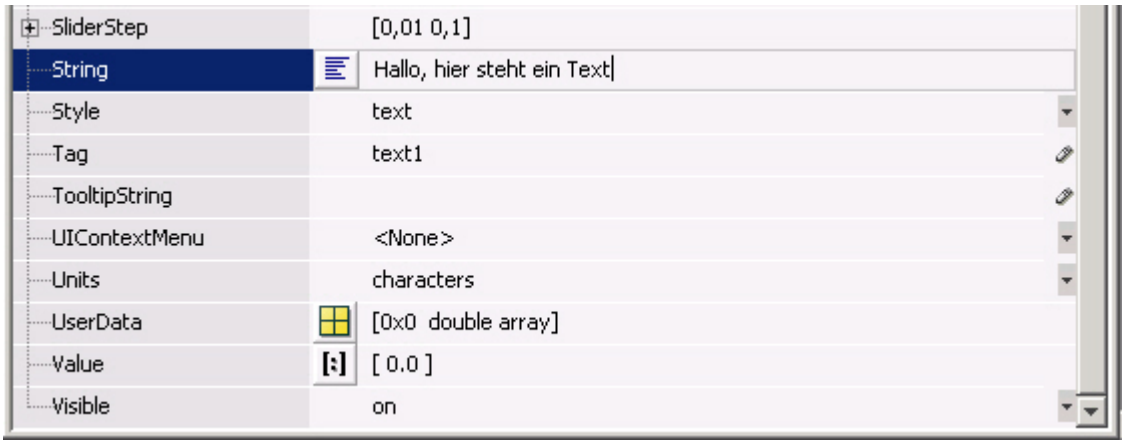
Daraufhin wird ein neues Fenster mit sämtlichen Eigenschaften des GUI-Elements, in diesem Fall dem statischen Textfeld mit dem Eintrag „Static Text“ geöffnet. Dieses Fenster ist sehr umfangreich.



Anhand der Überschrift erfahren wir immerhin schon einmal, dass der Name dieses GUI-Elements „`text1`“ vom Typ „`Static Text`“ ist.
Die Eigenschaften dieses Static-Text-Feldes können nun manuell geändert werden.



Den Eintrag, der als Text ausgegeben wird, ist unter der Eigenschaft „`String`“ zu finden. Dort steht: „`Static Text`“. Dieser Eintrag kann nun manuell geändert werden:



man klickt diesen Eintrag an und schreibt dann seinen Wunschtext hinein – hier also phantasielos „Hallo, hier steht ein Text“.

Man schließt den Inspector, vergrößert im Layout-Fenster noch ein wenig das GUI-Element und speichert das ganze ab.

Ruft man im Kommandofenster auf

```
>> Kap15_04_Gui2_StaticText
>>
```

dann erscheint

Noch eine Anmerkung:

Der Name des Static-Text-Elements ist „`text1`“ und wurde automatisch vom System vergeben. Falls uns dieser Name missfällt, kann er auch nachträglich geändert werden. Dazu wird wieder in GUIDE zu diesem Element der Property Inspector aufgerufen und in der Liste der Eigenschaften der Eintrag „`tag`“ gesucht – der hier noch „`text1`“ heißt. An dieser Stelle kann er manuell geändert werden.

Dies ist die erste Möglichkeit, einen Static-Text zu ändern – durch Aufruf des Property Inspectors.

Wir betrachten jetzt die zweite:

Eine Änderung zur Laufzeit ist möglich, indem man das Textfeld mit neuem Inhalt belegt. So kann dies beispielsweise geschehen, indem man beispielsweise in der Öffnungsfunktion `Kap15_04_Gui2_StaticText_OpeningFcn` das Textfeld über seinen Namen anspricht (hier also `text1`) und die Eigenschaft „String“ mit einem neuen Inhalt belegt (wir erinnern uns: die Öffnungsfunktion wird vom Gui-Hauptprogramm `gui_mainfcn` aufgerufen, bevor in einem Fenster der GUI auf dem Bildschirm erscheint). Das Kommando dazu ist bereits bekannt – es

ist die gute alte `set`-Funktion (wir müssten genauer sagen: die Methode `set`, die an das Objekt Static-Text-Feld geknüpft ist):

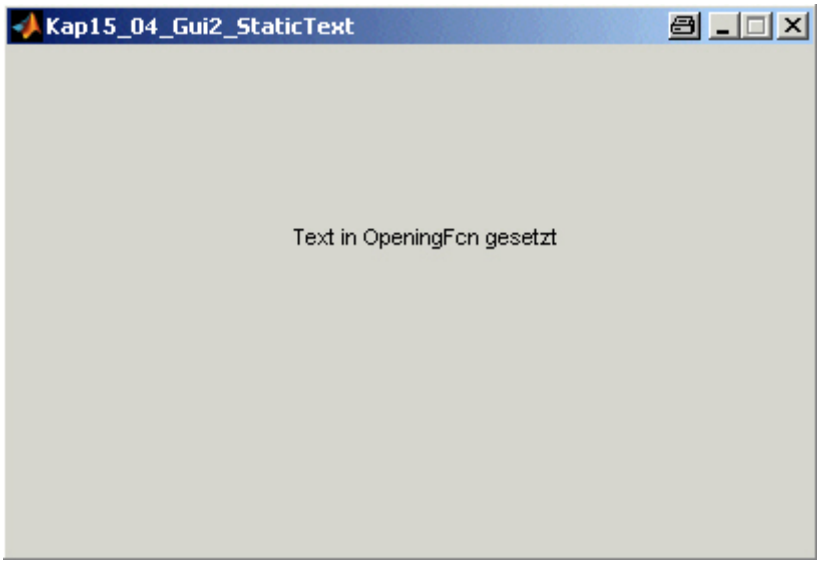
```
function Kap15_04_Gui2_StaticText_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_04_Gui2_StaticText (see
% VARARGIN)

% Choose default command line output for Kap15_04_Gui2_StaticText
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% im Textfeld von text1 die Eigenschaft 'String' setzen
set(handles.text1, 'String', 'Text in OpeningFcn gesetzt');
```

Der dazu notwendige Befehl ist also

```
set(handles.text1, 'String', 'Text in OpeningFcn gesetzt');
```

`set` wirkt auf die Struktur `handles`, von der ein Element das Static-Feld-Element `text1` ist. Dies geschieht analog zum Setzen des Textes mittels des Property-Inspectors. Der Unterschied besteht nur darin, dass dies zur Laufzeit des Programms geschieht, die Daten also gar nicht wie beim Property Inspector in einer Datei verwaltet und dann bei Programmstart abgerufen werden, sondern sie werden durch das laufende Programm geändert. Dies ist zu ersehen, wenn im Kommandofenster erneut aufgerufen wird:

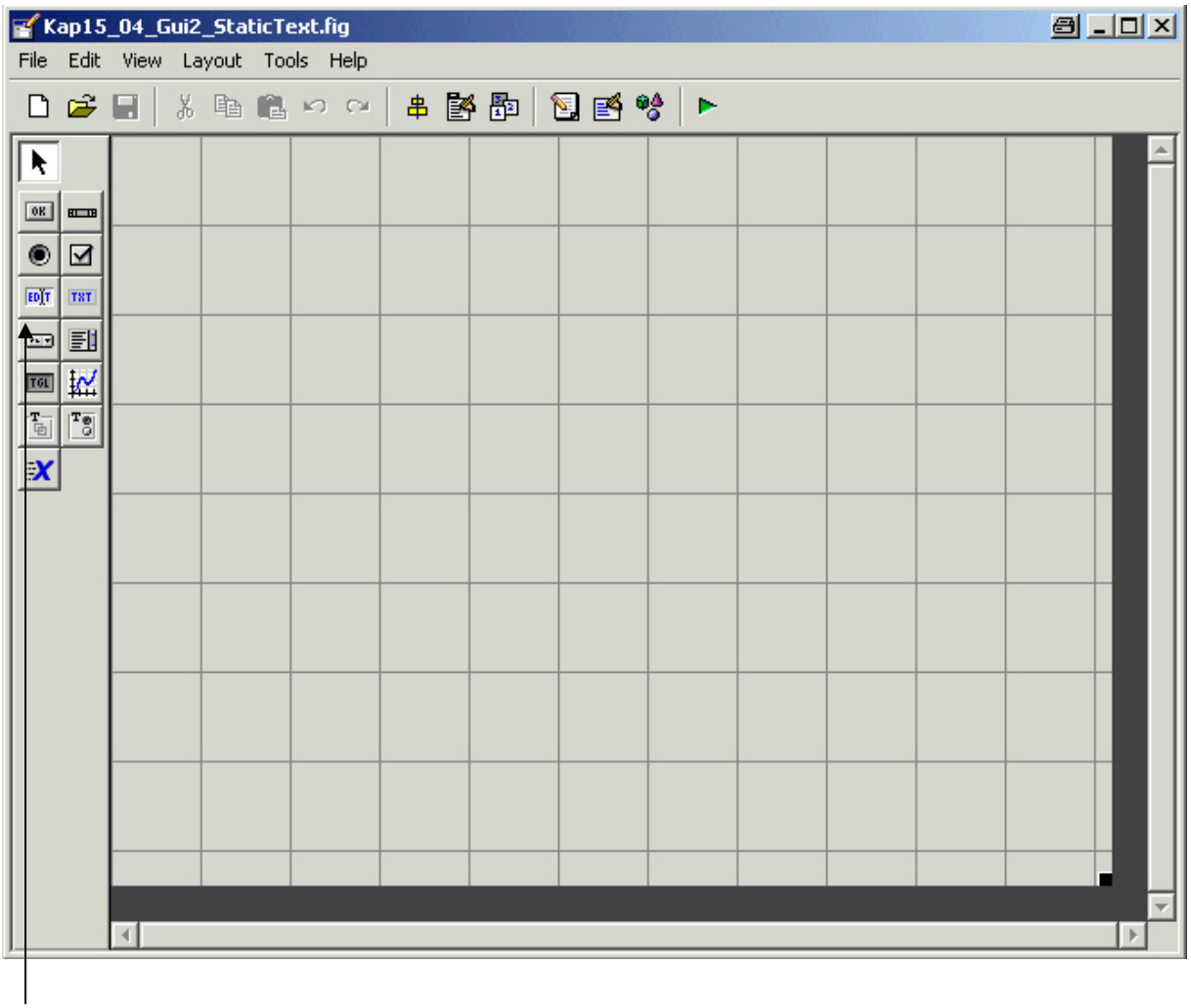
```
>> Kap15_04_Gui2_StaticText
>>
```



Der Text muß nicht unbedingt in der Öffnungsfunktion gesetzt werden – er kann auch anderswo eingelesen worden sein und wird dann diesem Part zugewiesen – beispielsweise durch eine Rückruf-Funktion, die bezüglich eines (hier noch nicht erstellten) weiteren GUI-Elements implementiert ist.

Texteingabefeld in einem GUI

Wir wollen ein Feld im GUI erzeugen, indem nach Aufruf des GUI's nachträglich vom Anwender durch Anklicken über die Tastatur ein Text eingegeben werden kann. Ein solches Element heißt in GUIDE „Edit Text"



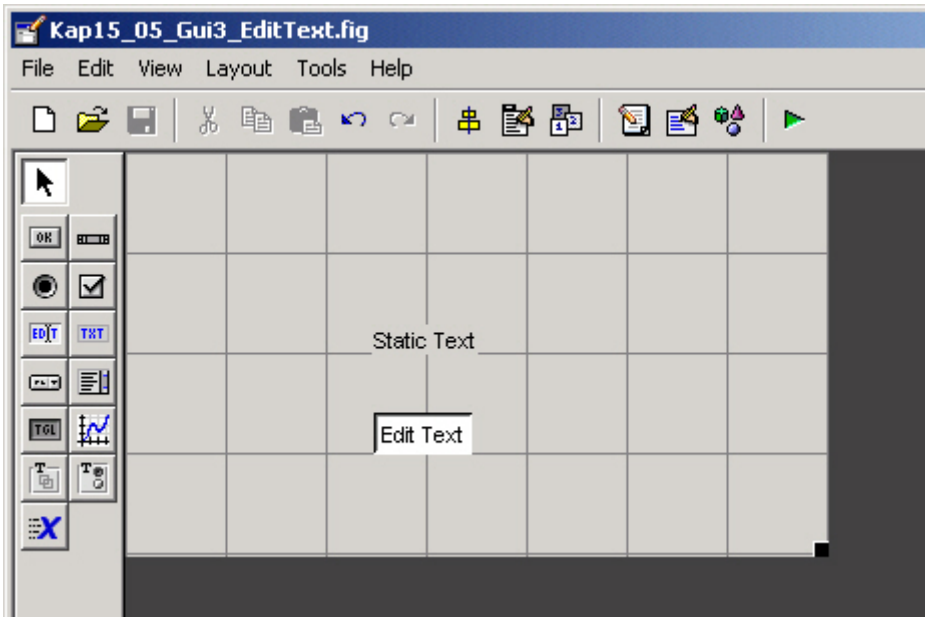
Edit Text

Das dazugehörige Auswahlelement befindet sich in der Auswahlpalette links neben dem für „Static Text", also in der ersten Spalte.

Wir erzeugen einen neuen GUI, in dem wir wie beim vergangenen ein Static-Text-Feld ablegen, dann aber noch zusätzlich ein Edit-Text-Feld darunter legen.

Aus dem Property-Inspector entnehmen wir, dass der Name des Static-Text-Feldes `text1` ist und der des Edit-Text-Feldes edit1. Diese Namen benötigen wir nachher, um ihnen Werte zuweisen zu können.

Das ganze speichern wir unter „`Kap15_05_Gui3_EditText.fig`“ ab:



Gleichzeitig ist ein m-File „`Kap15_03_Gui3_EditText.m`“ angelegt worden. Er enthält neben der Öffnungs- und Ausgabefunktion auch eine Rückruf-Funktion und Create-Funktion:

```
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Die Create-Funktion wird beim Start aufgerufen, bevor das Fenster geöffnet wird. Sie gibt dem Anwender die Möglichkeit, speziell dieses Feld weiter zu editieren. Wie aus dem Text zu ersehen, wird hier die Hintergrundfarbe „white“ gesetzt. Dies ist eine durchaus sinnvolle Standardvorbesetzung. Wir lassen sie daher hier unverändert.

Die Rückruf-Funktion `edit1_Callback` wird immer dann aufgerufen, wenn der Anwender mit der Maus in das Eingabefeld fährt, dort etwas über die Tastatur eintippt und mit der Enter-Taste abschließt (also nicht bereits während der Eingabe).

Den eingegebenen Text kann man mit der `get`-Funktion in einer Variablen speichern:

```
txt = get(handles.edit1, 'String');
```

Und entsprechend dem Static-Text-Feld wieder zuweisen:

```
set(handles.text1, 'String', txt);
```

Damit hat die Rückruf-Funktion folgendes komplettes Aussehen:

```
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1
%        as a double
txt = get(handles.edit1, 'String');
set(handles.text1, 'String', txt);
```

Weil’s so schön war, wurde auch noch ein bißchen die Größe der Felder und der Eingangstext manipuliert. Dies geschieht bekanntlich in der Öffungsfunktion

```
% --- Executes just before Kap15_05_Gui3_EditText is made visible.
function Kap15_05_Gui3_EditText_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_05_Gui3_EditText (see
%            VARARGIN)

% Choose default command line output for Kap15_05_Gui3_EditText
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% Abmessungen vergrößern
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
% Eingabetext zu Beginn
set(handles.edit1, 'String', 'Gib hier was ein');
```

```
% UIWAIT makes Kap15_05_Gui3_EditText wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

Mittels der bekannten `set`-Funktion wurden die Position und die Abmessungen beider Textfelder festgelegt;

der Vektor hat folgende Bedeutung: [x, y, Breite, Höhe]

ebenso wurde für die Eröffnung ein Anfangstext gewählt.

Ruft man jetzt im Kommandofenster auf

```
>> h = Kap15_05_Gui3_EditText
>>
```

erscheint das folgende Fensterchen:

Rückgabewert ist hier, da über `varargout` keine Ausgabe festgelegt wurde, die Nummer auf diesen Handle:

```
h =

  157.0022

>>
```

Dies ist hier allerdings von nebensächlicher Bedeutung.

Gibt man nun in das Textfeld etwas ein und schließt diesen mit der Eingabe-Taste ab (und zwar erst dann!), dann wird der Eingabetext dem Static-Text-Feld übergeben und angezeigt:

Das Hin- und Her sich gegenseitig aufrufender Funktionen ist damit sehr viel aufwändiger geworden.

Bei Öffnen des GUI's geschieht folgendes:

1. im Kommandofenster wird der Befehl
   `h = Kap15_05_Gui3_EditText`
   eingegeben.
2. Es wird aufgerufen
   `varargout = Kap15_05_Gui3_EditText(varargin)`
   Übergabeparameter `varargin` ist leer
   Rückgabeparameter `varargout{1}` = `h`
3. Es wird `gui_State` gesetzt
4. Aufruf der Hauptfunktion `gui_mainfcn` mit Übergabeparameter `gui_State`.
   a. Neben diesem Übergabeparameter für die Handhabung des GUI's soll das Eingabefenster `edit1` erzeugt werden über die Funktion `edit1_CreateFcn`. Daher wird gesetzt
   `varargin{1}` = „`edit1_CreateFcn`".
   b. Aufruf von `varargout = Kap15_05_Gui3_EditText(varargin)` mit `varargin{1}` = "`edit1_CreateFcn`".
      i. Es ist `varargin{1}` ein String. Also ist in
      ```
      if nargin && ischar(varargin{1})
         gui_State.gui_Callback = str2func(varargin{1});
      end
      ```
      die `if`-Bedingung erfüllt, d.h. es wird gesetzt
      `gui_State.gui_Callback` = „`edit1_CreateFcn`"
      ii. Aufruf von `gui_mainfcn` mit Übergabeparameter `gui_State`
         1. Aufruf von `edit1_CreateFcn`

2. Abarbeiten von `edit1_CreateFcn`; setzen der dort vorgegebenen Eigenschaften für das Eingabefenster `edit1`.
3. Beenden von `edit1_CreateFcn` und Rücksprung nach `gui_mainfcn`.

iii. `gui_mainfcn` führt keine weiteren Anweisungen aus, sondern wird beendet. Rücksprung nach `Kap15_05_Gui3_EditText`

c. `Kap15_05_Gui3_EditText` führt keine weiteren Anweisungen aus, sondern wird beendet. Rücksprung nach `gui_mainfcn`.

d. Aufruf von `Kap15_05_Gui3_EditText_OpeningFcn(hObject, eventdata, handles, varargin)`

i. Abmessungen für den GUI werden gesetzt

ii. Eingabefenster wird mit „Gib hier was ein“ vorbesetzt

iii. Rücksprung nach `gui_mainfcn`

e. Aufruf von `varargout = Kap15_05_Gui3_EditText_OutputFcn(hObject, eventdata, handles)`

i. `varargout{1}` wird auf den Handle zu dem GUI gesetzt

ii. Rücksprung nach `gui_mainfcn`

f. `gui_mainfcn` wird beendet mit Rückgabewert `varargout{1}` = Handlenummer. Rücksprung nach `varargout = Kap15_05_Gui3_EditText`

5. `Kap15_05_Gui3_EditText` wird beendet mit Rückgabewert `varargout{1}` = Handlenummer. Rücksprung auf Kommandoebene
6. Im Kommandofenster wird die Handlenummer ausgegeben.

Im Schaubild:

Abbildung 3

Abfolge der Funktionen nach Aufruf des Befehls „h = Kap15_05_Gui3_EditText“ im Kommandofenster

Nicht ganz so wüst sieht es aus, wenn man in das Texteingabefeld etwas einträgt und den Eintrag mit der Eingabetaste abschließt:

1. Eingabe in das Eingabefenster „ich geb was ein“ und mit Eingabe-Taste abschließen.
2. Aufruf von `varargout = Kap15_05_Gui3_EditText(varargin)`
   mit `varargout` = leer und `varargin{1}` = „`edit1_Callback`“
3. Damit ist die `if`-Bedingung von

   ```
   if nargin && ischar(varargin{1})
      gui_State.gui_Callback = str2func(varargin{1});
   end
   ```

   erfüllt; es wird also neben den anderen (Standard-)werten `gui_State` besetzt:
   `gui_State.gui_Callback` = „`edit1_Callback`”.
4. `gui_mainfcn` wird mit Übergabewert `gui_State` aufgerufen
   a. `gui_mainfcn` ruft `edit1_Callback(hObject, eventdata, handles)` auf.
      i. Übernimmt den Eintrag aus dem Eingabefenster `edit1` und weist ihn dem Textfeld `text1` zu.
      ii. `Edit1_Callback` ist beendet. Rücksprung nach `gui_mainfcn`
   b. Der GUI wird aktualisiert – es steht jetzt der gleiche Text über dem Eingabefeld wie in demselben: „ich geb was ein“.
   c. `gui_mainfcn` ist beendet. Rücksprung nach `Kap15_05_Gui3_EditText`
5. `Kap15_05_Gui3_EditText` wird beendet. Rücksprung auf die Kommandoebene.

Die Abfolge ist also analog zu der des ersten GUI’s nach Betätigen des Kommandoknopfes „Kreis zeichnen“.

Die Abfolge noch einmal graphisch dargestellt:

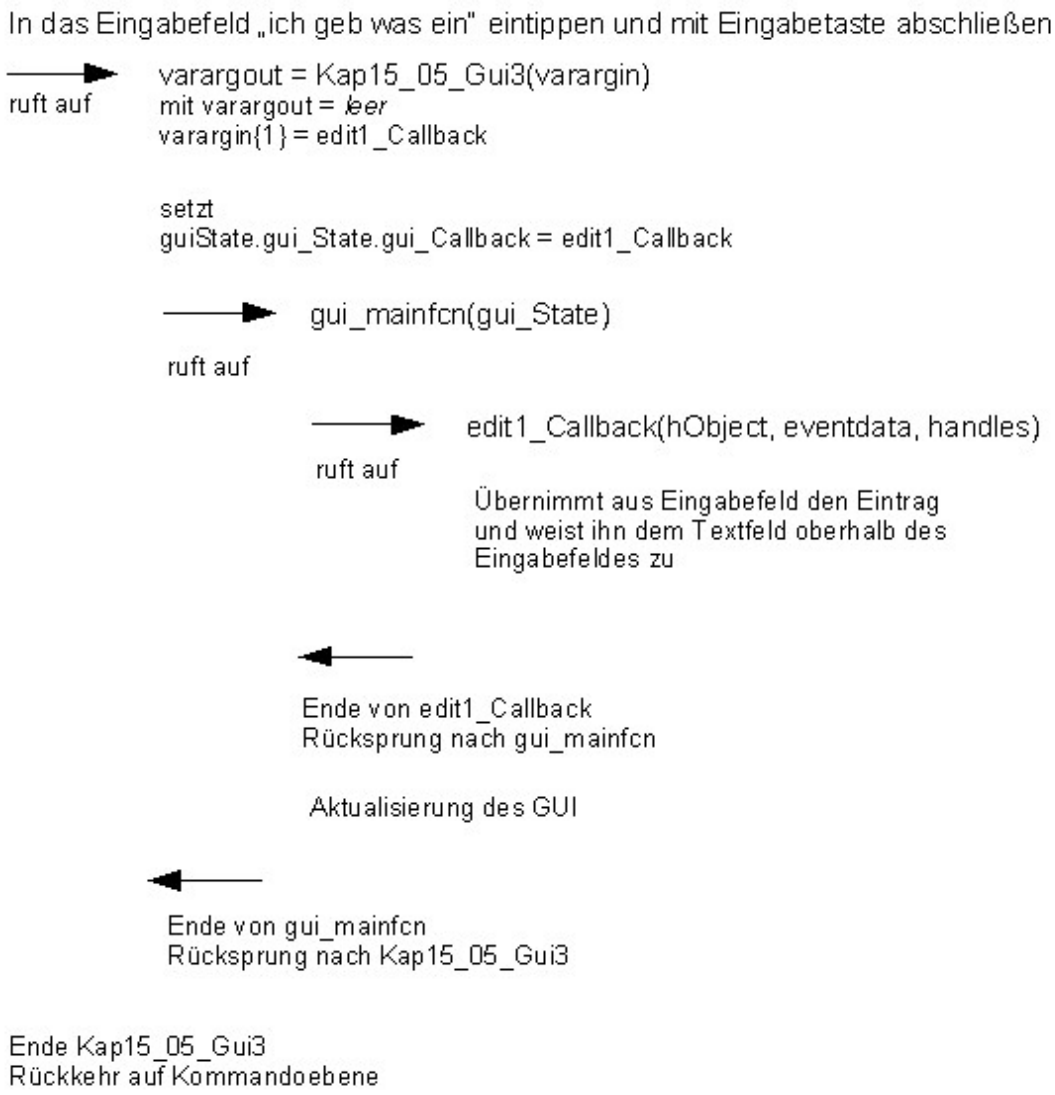


Abbildung 4 Abfolge der Funktionen nach Eingabe eines Textes in das Eingabefeld

Somit ist es also möglich, Text einzugeben und diese zu verarbeiten. Allerdings können derartig gewonnene Daten auf diese Weise nicht über den Funktionswert des GUI zurückgegeben werden. Mit ein paar zusätzlichen Kommandos kann jedoch auch dies realisiert werden. Dies ist Gegenstand des nächsten Abschnittes.

Der GUI-Rückgabewert

Ziel ist, einen über einen GUI gewonnene Daten im Rückgabewert der Funktion zu speichern, die den GUI aufruft. Wir werden dies anhand eines Textes illustrieren, den wir über ein Eingabefeld im GUI eintippen und ihn uns ins Kommandofenster zurückgeben lassen.

Im Prinzip haben wir diesen bereits einmal angesprochen bei der Betrachtung der Rückgabefunktion. Wie vielleicht noch erinnerlich, wurde in der Rückgabefunktion des ersten

GUI's, in `Kap15_03_Gui1_OutputFcn`, der Variablen `varargout{1}` eine Zeichenkette zugewiesen:

```
function varargout = Kap15_03_Gui1_OutputFcn(hObject, eventdata, handles)

varargout{1} = 'Mein GUI';
```

Beim Aufruf von `Kap15_03_Gui1` wird dieser Text als Funktionswert an das aufrufende Programm (in diesem Fall das Kommandofenster) zurückgegeben

```
>> h=Kap15_03_Gui1

h =

Mein GUI

>>
```

Im Grunde können wir hier dasselbe Vorgehen wählen:

In `XX_OutputFcn` wird also `varargout{1}` eine Zeichenkette zugewiesen, die „irgendwie" den Inhalt aus dem Texteingabefeld des GUI erhält, nachdem der Anwender dort etwas eingetippt und mit der Enter-Taste abgeschlossen hat. Die dafür zuständige Funktion ist `XX_edit(hObjeckt, eventdata, handles).`

In der Ausgabe-Funktion `XX_edit1(hObject, eventdata, handles)` lassen wir mit `get` den Text aus dem Eingabefeld des GUI auslesen, den wir mit `set` dem Textfeld zuweisen. Wie aber bekommt man den Wert des Textes jetzt aus `XX_edit1` heraus, um auf ihn in einer anderen Funktion wieder zugreifen zu können?

Jetzt kommt endlich der Parameter `handles` ins Spiel, der in (fast) allen Funktionen übergeben wird. Diesen erweitern wir um das Element `mytext` und weisen ihm den Inhalt des ausgelesenen Textfeldes zu. Danach muß, da die Struktur von `handles` verändert wurde, die Änderung dem System mitgeteilt werden. Dies geschieht mittels des Befehls

```
guidata(hObject, handles);
```

Also:

Wir erzeugen also mittels GUIDE erneut einen GUI, bauen in diesen ein Textausgabe- und Texteingabefeld ein und speichern diesen unter „Kap15_06_Gui4Entwurf" ab.

Die dabei mit erzeugte Datei „`Kap15_06_Gui4Entwurf.m`" ändern wir wie folgt ab:

```
function varargout = Kap15_06_Gui4Entwurf(varargin)
% KAP15_06_GUI4ENTWURF M-file for Kap15_06_Gui4Entwurf.fig
```

```
%      KAP15_06_GUI4ENTWURF, by itself, creates a new KAP15_06_GUI4ENTWURF
%      or raises the existing singleton*.
%
%      H = KAP15_06_GUI4ENTWURF returns the handle to a new
%      KAP15_06_GUI4ENTWURF or the handle to the existing singleton*.
%
%      KAP15_06_GUI4ENTWURF('CALLBACK',hObject,eventData,handles,...) calls
%      the local function named CALLBACK in KAP15_06_GUI4ENTWURF.M with the
%      given input arguments.
%
%      KAP15_06_GUI4ENTWURF('Property','Value',...) creates a new
%      KAP15_06_GUI4ENTWURF or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_06_Gui4Entwurf_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to
%      Kap15_06_Gui4Entwurf_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_06_Gui4Entwurf

% Last Modified by GUIDE v2.5 19-Jul-2007 09:40:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_06_Gui4Entwurf_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_06_Gui4Entwurf_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_06_Gui4Entwurf is made visible.
function Kap15_06_Gui4Entwurf_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_06_Gui4Entwurf (see VARARGIN)

% Choose default command line output for Kap15_06_Gui4Entwurf
handles.output = hObject;
handles.mytext = 'noch nix da';
% Abmessungen vergrößern
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
% Eingabetext zu Beginn
```

```
set(handles.edit1, 'String', 'Gib hier was ein');

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_06_Gui4Entwurf wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_06_Gui4Entwurf_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.mytext;




function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double
txt = get(handles.edit1, 'String');
set(handles.text1, 'String', txt);
% Erweiterung der Struktur handles zur Aufnahme des Textes, um diesen
% an die Ausgabe-Funktion übergeben zu können
handles.mytext = txt;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

In der Funktion `Kap15_06_Gui4Entwurf_OpeningFcn` wurde noch gesetzt:

Die Startwerte für den GUI:

```
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
set(handles.edit1, 'String', 'Gib hier was ein');
```

Und das neue Element von handles mit dem Namen `mytext` wurde vorbesetzt:

```
handles.mytext = 'noch nix da';
```

Die Änderungen an der Struktur von `handles` wird dem System mitgeteilt.

```
guidata(hObject, handles);
```

Die Rückgabefunktion `varargout = Kap15_06_Gui4Entwurf_OutputFcn(hObject, eventdata, handles)`
erhält auch den das neue Element handles.mytext als Rückgabewert zugewiesen:

```
varargout{1} = handles.mytext;
```

Die Rückruf-Funktion `edit1_CreateFcn(hObject, eventdata, handles)`
liest auch brav den Inhalt aus dem Texteingabefeld aus und weist ihn `handles.mytext` zu – und aktualisiert die Struktur des `handles`:

```
txt = get(handles.edit1, 'String');
handles.mytext = txt;
guidata(hObject, handles);
```

Die Vorbesetzungen funktionieren korrekt. Dies beweist nach Aufruf des Befehls

```
>> h = Kap15_06_Gui4Entwurf
```

im Kommandofenster erscheint – wie vorgegeben – der GUI:

es erscheint im Kommandofenster die Abfolge:

```
h =

noch nix da

>>
```

Dies ist in der Tat die Vorbesetzung von `handles.mytext`, die in

```
varargout = Kap15_06_Gui4Entwurf_OutputFcn(hObject, eventdata, handles)
```

dem Rückgabewert `varargout` zugewiesen wurde:

```
varargout{1} = handles.mytext;
```

Die Abfolge der Funktionen ist bereits besprochen und findet auch hier wieder statt:
Über das Kommando „`h = Kap15_06_Gui4Entwurf`“ wird die Funktion
`Kap15_06_Gui4Entwurf`
aufgerufen; diese ruft über `gui_mainfcn` erst einmal erneut sich selbst auf, um letztendlich (die hier nicht weiter interessierende) Funktion `edit1_CreateFcn` abzuarbeiten, dann aber (wieder über `gui_mainfcn`) zunächst `Kap15_06_Gui4Entwurf_OpeningFcn` und dann `Kap15_06_Gui4Entwurf_OutputFcn` aufzurufen; damit sind dann Texteingabefeld und `handles.mytext` vorbesetzt. Dann erst wird der GUI auf den Bildschirm gebracht.

So weit so gut.

Jetzt tippen wir in das Eingabefeld des GUI den Text „Neueingabe“ ein und Betätigen die Enter-Taste. Wie gewollt, wird dank der Rückruf-Funktion `edit1_Callback` der Inhalt des Eingabefeldes übernommen und dem Textfeld zugewiesen:

Doch im Kommandofenster zeigt sich nichts – keine Rückgabe des eingegebenen Textes „Neueingabe“.
Dies ist nicht verwunderlich, denn die Abfolge des Funktionenaufrufs sieht ja gar keinen Aufruf der Rückgabefunktion vor:

durch Texteingabe und Betätigen der Enter-Taste wird die Rückruf-Funktion `edit1_Callback` aufgerufen (genauer: Aufruf `Kap15_06_Gui4Entwurf`, dann `gui_mainfcn` und von dieser aus dann `edit1_Callback`); ein Aufruf von `Kap15_06_Gui4Entwurf_OutputFcn` und damit Setzen des neu eingelesenen Werte, der in `handles.mytext` steht, ist nicht vorgesehen.

Damit zeigt sich der Nachteil dieser Anordnung:
`Kap15_06_Gui4Entwurf_OutputFcn` wird nur bei Öffnen des GUI's aufgerufen und damit `varargout` mit `handles.mytext` belegt. Dies geschieht, bevor der Anwender eine Chance hat, den von ihm gewünschten Text in das Eingabefeld zu tippen.
Daher muß folgendes realisiert werden:

- die Ausgabe des Funktionswertes so lange zurückgehalten werden, bis der Dialog beendet ist
- der in das Eingabe-Fenster eingegebene Text muß der Ausgabe-Funktion übergeben werden, damit diese das entsprechende Element von varargout setzen kann.

Das letzte Problem haben wir bereits durch Einführung eines weiteren Elementes in der Struktur `handles` (das Element `mytext`) in Angriff genommen.
Das erste Problem ist mit MATLAB auch leicht zu lösen:
Man aktiviert in der Öffnungsfunktion den Aufruf `uiwait`.
Dann wartet das Hauptprogramm nämlich so lange, bis der GUI geschlossen wird oder ein Aufruf von `uiresume` erfolgt.

Die Funktionen uiwait und uiresume

Diese beiden Funktionen haben folgende Aufgabe:
`uiwait` blockiert die Ausführung so lange, bis der Befehl `uiresume` erfolgt oder das aktuelle Programm beendet wird. Als Argument kann der handle auf den aktuellen GUI genommen werden – dann ist nur die Ausführung bzgl. dieses GUI's blockiert (in unserem Fall also `figure1`).

In unserem Fall modifiziert das vorangegangene Programm wie folgt:
An das Ende der Öffnungsfunktion setzen wir den Aufruf

```
uiwait(handles.figure1)
```

d.h. wir aktivieren diese Funktion einfach, denn der Befehl wurde dort bereits bei Erzeugung des GUI's automatisch mit eingefügt, allerdings durch vorangestelltes Kommentarzeichen deaktiviert.

Die modifizierte Version ist unter `Kap15_07_Gui4Erweitert` abgelegt:

```
function varargout = Kap15_07_Gui4Erweitert(varargin)
% KAP15_07_GUI4ERWEITERT M-file for Kap15_07_Gui4Erweitert.fig
%      KAP15_07_GUI4ERWEITERT, by itself, creates a new
%      KAP15_07_GUI4ERWEITERT or raises the existing singleton*.
%
%      H = KAP15_07_GUI4ERWEITERT returns the handle to a new
%      KAP15_07_GUI4ERWEITERT or the handle to the existing singleton*.
%
%      KAP15_07_GUI4ERWEITERT('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in KAP15_07_GUI4ERWEITERT.M
%      with the given input arguments.
%
%      KAP15_07_GUI4ERWEITERT('Property','Value',...) creates a new
%      KAP15_07_GUI4ERWEITERT or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_07_Gui4Erweitert_OpeningFunction gets called.  An unrecognized
%      property name or invalid value makes property application stop.  All
%      inputs are passed to Kap15_07_Gui4Erweitert_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_07_Gui4Erweitert

% Last Modified by GUIDE v2.5 19-Jul-2007 12:35:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_07_Gui4Erweitert_OpeningFcn,
...
                   'gui_OutputFcn',  @Kap15_07_Gui4Erweitert_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_07_Gui4Erweitert is made visible.
function Kap15_07_Gui4Erweitert_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_07_Gui4Erweitert (see
%            VARARGIN)

% Choose default command line output for Kap15_07_Gui4Erweitert
handles.output = hObject;
handles.mytext = 'noch nix da';
% Abmessungen vergrößern
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
% Eingabetext zu Beginn
set(handles.edit1, 'String', 'Gib hier was ein');

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_07_Gui4Erweitert wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_07_Gui4Erweitert_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.mytext;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double
txt = get(handles.edit1, 'String');
set(handles.text1, 'String', txt);
% Erweiterung der Struktur handles zur Aufnahme des Textes, um diesen
% an die Ausgabe-Funktion übergeben zu können
handles.mytext = txt;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Wie zu ersehen, ist die (bis jetzt) einzige vorgenommene Modifikation der Zusatz `uiwait()` am Ende der Funktion `Kap15_07_Gui4Erweitert_OpeningFcn`.

Der Aufruf im Kommandofenster

```
h = Kap15_07_Gui4Erweitert
```

bringt in der Tat den GUI mit den bereits bekannten Vorbesetzungen auf den Bildschirm:

im Gegensatz zur vorangegangenen Version wird hier nichts im Kommandofenster ausgegeben – die Ausgabe wird zurückgehalten.

In der Tat ist bei Öffnen des GUI's die Abfolge der Funktionen folgende:

1) Aufruf im Kommandofenster: `h = Kap15_07_Gui4Erweitert`
2) Aufruf `varargout = Kap15_07_Gui4Erweitert(varargin)`
   mit `varargin` = *leer* und einem Rückgabeparameter `h` = `varargout{1}`
   a) Aufruf `varargout` = `gui_mainfcn(gui_State)`
   b) Wegen des Texteingabefeldes wird gesetzt
      `varargin{1}` = „`edit1_CreateFcn`"
   c) Aufruf von `varargout = Kap15_07_Gui4Erweitert(varargin)` mit
      `varargin{1}` = „`edit1_CreateFcn`"
      i) Damit wird
         `gui_State.gui_Callback` = `str2func(varargin{1})`
         = „`edit1_CreateFcn`";
      ii) Aufruf von `varargout = gui_mainfcn(gui_State)`;
         (1) Aufruf von `edit1_CreateFcn`
            (a) Abarbeiten von `edit1_CreateFcn` und Rücksprung nach `gui_mainfcn`

iii) Beenden von `gui_mainfcn` und Rücksprung nach `varargout = Kap15_07_Gui4Erweiterung(varargin)`

d) Wieder in `Kap15_07_Gui4Erweiterung`. Es ist hier `gui_State.gui_Callback` = []

e) Aufruf von `varargout = gui_mainfcn(gui_State)`

i) Aufruf von `Kap15_07_Gui4Erweitert_OpeningFcn(hObject, ...)`

(1) Zuweisung `handles.mytext` = „`noch nix da`“

(2) Aufruf „`uiwait`“

Der Befehl von „`uiwait`“ in der Öffnungsfunktion `Kap15_07_Gui4Erweitert_OpeningFcn(hObject, ...)` hat als Resultat eine Unterbrechung der Programmausführung zur Folge. Damit erfolgt insbesondere kein Rücksprung nach `gui_mainfcn` und anschließend nach `Kap15_07_Gui4Erweitert`. Das Programm wird nicht mit der Übergabe des Parameters `handles.mytext` beendet, sondern gibt an das Betriebssystem eine vorläufige Endemeldung. Insbesondere erfolgt keine Ausgabe von `h` im Kommandofenster.

Jetzt tragen wir über die Tastatur den Eintrag „Neueingabe“ in das Texteingabefeld ein und schließen diesen Eintrag mit der Enter-Taste ab.

Tatsächlich wird die Eingabe ins Textfeld übernommen – es muß also zu einem Aufruf der Rückruf-Funktion `edit1_Callback` gekommen sein.

Doch im Kommandofenster erscheint immer noch keine Ausgabe von h = „Neueingabe“.
Die Aufrufe der Funktionen sind jetzt wie folgt:

1) Eingabe über die Tastatur des Texte „Neueingabe“ und Abschluß mit Enter-Taste
2) Aufruf der Funktion `varargout = Kap15_07_Gui4Erweitert(varargin)`
   mit `varargout` = *leer*(!)
   und `varargin{1}` = „edit1_Callback“ (es ist `varargout` = *leer*, da `edit1_Callback` keine Rückgabewerte hat).
3) Gesetzt wird `gui_State.gui_Callback` = „edit1_Callback“
   a) Aufruf von `gui_mainfcn` mit Übergabeparameter `gui_State`.
      i) Aufruf von `edit1_Callback`
      ii) Es wird der eingebene Text “Neueingabe” ausgelesen und dem Textfeld zugewiesen
      iii) `handles.mytext` = „Neueingabe“ wird gesetzt
      iv) Damit ist `edit1_Callback` abgearbeitet. Rücksprung nach `gui_mainfcn`
   b) `uiwait` verhindert Aufruf der Rückgabe-Funktion `Kap15_07_Gui4Erweitert_OutputFcn`. `gui_mainfcn` ist abgearbeitet. Rücksprung nach `Kap15_07_Gui4Erweitert`.
4) Beenden von `Kap15_07_Gui4Erweitert`.
5) Rückkehr auf Kommandoebene.

Wie zu ersehen, wird also die Rückgabefunktion, in der letztendlich

```
varargout{1} = handles.mytext;
```

gesetzt wird, gar nicht ausgeführt.
Tatsächlich ist trotz Aktualisierungsbefehl `guidata` das neue Element `handles.mytext` gar nicht für die Ausgabe registriert worden. Schließen wir nämlich den GUI, gibt’s einen Aufschlag, weil die geänderte Struktur nicht an die aufrufende Funktion zurückgegeben worden ist.

Des Problems Lösung liegt in der Anwendung des Befehls `uiresume`. Dann wird die Blockade gelöst, die beim Aufruf von `uiwait` entstand.

Wo positioniert man `uiresume`?
Packen wir daher `uiresume` probeweise ans Ende der Rückruf-Funktion `edit1_Callback(hObject, eventdata, handles)`. Dann hat das Programm mittlerweile das Aussehen

```
function varargout = Kap15_07_Gui4Erweitert(varargin)
% KAP15_07_GUI4ERWEITERT M-file for Kap15_07_Gui4Erweitert.fig
%      KAP15_07_GUI4ERWEITERT, by itself, creates a new
%      KAP15_07_GUI4ERWEITERT or raises the existing singleton*.
%
%      H = KAP15_07_GUI4ERWEITERT returns the handle to a new
%      KAP15_07_GUI4ERWEITERT or the handle to the existing singleton*.
%
%      KAP15_07_GUI4ERWEITERT('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in KAP15_07_GUI4ERWEITERT.M
%      with the given input arguments.
%
%      KAP15_07_GUI4ERWEITERT('Property','Value',...) creates a new
%      KAP15_07_GUI4ERWEITERT or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_07_Gui4Erweitert_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to
%      Kap15_07_Gui4Erweitert_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_07_Gui4Erweitert

% Last Modified by GUIDE v2.5 19-Jul-2007 12:35:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_07_Gui4Erweitert_OpeningFcn,
...
                   'gui_OutputFcn',  @Kap15_07_Gui4Erweitert_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_07_Gui4Erweitert is made visible.
function Kap15_07_Gui4Erweitert_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_07_Gui4Erweitert (see
%            VARARGIN)

% Choose default command line output for Kap15_07_Gui4Erweitert
handles.output = hObject;
handles.mytext = 'noch nix da';
```

```
% Abmessungen vergrößern
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
% Eingabetext zu Beginn
set(handles.edit1, 'String', 'Gib hier was ein');

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_07_Gui4Erweitert wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_07_Gui4Erweitert_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.mytext;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double
txt = get(handles.edit1, 'String');
set(handles.text1, 'String', txt);
% Erweiterung der Struktur handles zur Aufnahme des Textes, um diesen
% an die Ausgabe-Funktion übergeben zu können
handles.mytext = txt;
guidata(hObject, handles);
uiresume(handles.figure1);

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Gibt man also wieder in das Kommandofenster ein

```
>> h = Kap15_07_Gui4Erweitert
```

So erscheint der GUI.

Eintippen von „Neueingabe“ in das Eingabefenster bringt erfolgreich nicht nur die Übernahme dieses Textes in das Textfeld, sondern auch die Ausgabe ins Kommandofenster

```
h =
Neueingabe
>>
```

Schließt man den GUI, geht auch das problemlos – es wird keine weitere Funktion aufgerufen.

Jetzt ist bei Aufruf von
`h = Kap15_07_Gui4Erweitert`
der Ablauf der folgende:

1) Aufruf `h = Kap15_07_Gui4Erweitert` im Kommandofenster
2) Aufruf von `varargout = Kap15_07_Gui4Erweitert(varargin)`
   mit `varargout{1}` = `h`
   und `varargin` = *leer*
3) Aufruf und Abarbeiten von `edit1_CreateFcn`.
4) Aufruf von `gui_mainfcn`
   a) Aufruf von `Kap15_07_Gui4Erweitert_OpeningFcn(hObject, eventdata, handles, varargin)`
      i) Setzen der Abmessungen, `handles.mytext` = „`noch nix da`“
      ii) Aufruf „`uiwait(handles.figure1)`“
      iii) Programm unterbrochen
      iv) Eingabe ins Texteingabefeld „`Neueingabe`“ und mit Enter-Taste abgeschlossen.
      v) Aufruf `Kap15_07_Gui4Erweitert`
         mit `varargout` = *leer*
         und `varargin{1}` = „`edit1_Callback`“
         (1) Aufruf `gui_mainfcn`
            (a) Aufruf `edit1_Callback(hObject, eventdata, handles)`
            (b) Auslesen des Textes aus dem Texteingabefeld und Zuweisung an `handles.mytext`
            (c) Aufruf `uiresume`
            (d) Beenden von `edit1_Callback` und Rücksprung nach `gui_mainfcn`
         (2) Beenden von `gui_mainfcn` und Rücksprung nach `Kap15_07_Gui4Erweitert`

Beenden von `Kap15_07_Gui4Erweitert`

vi) Wegen „`uiresume`“ Wiedereinstieg in `Kap15_07_Gui4Erweitert_OpeningFcn`

vii) Es ist jetzt `handles.mytext` noch immer „`noch immer nix da`“

viii) Beenden von von `Kap15_07_Gui4Erweitert_OpeningFcn` und Rücksprung nach `gui_mainfcn`.

b) `gui_mainfcn` aktualisiert (endlich) die Struktur `handles`

(Befehlt in gui_mainfcn: `gui_Handles = guidata(gui_hFigure);`)

c) Aufruf von `varargout = Kap15_07_Gui4Erweitert_OutputFcn(hObject, eventdata, handles)`

mit dem aktualisierten `handles.mytext` = “`Neueingabe`”

i) Zuweisung `varargout{1}` = `handles.mytext`

ii) Beenden von `Kap15_07_Gui4Erweitert_OutputFcn` und Rücksprung nach `gui_mainfcn`

d) Beenden von `gui_mainfcn` und Rücksprung nach `Kap15_07_Gui4Erweitert`

5) Beenden von `Kap15_07_Gui4Erweitert` mit `h` = `varargout{1}` = „`Neueingabe`“

6) Ausgabe von `h` im Kommandofenster.

Jetzt endlich hat es also geklappt! Der ins Texteingabefeld eingegebene Text ist im Kommandofenster wieder ausgegeben worden.

Etwas trübt allerdings die Freude.

Es wird nur beim ersten Mal der Text aus dem Eingabefeld übernommen und `h` zugewiesen:

Gibt man erneut einen weiteren Text in das Eingabefeld, wird zwar das Textfeld angepasst, doch der neue Text wird nicht (mehr) in das Kommandofenster geschrieben:

Dies ist auch nicht weiter verwunderlich:
Eine erneute Texteingabe in den noch immer geöffneten GUI ruft zwar erneut die Rückruf-Funktion `edit1_Callback` auf, die den neu eingegebenen Text aus dem Texteingabefeld der Variablen `handles.mytext` zuweist, doch erfolgt kein Aufruf der Ausgabe-Funktion `Kap15_07_Gui4Erweitert_OutputFcn`, in der `handles.mytext` dem (ohnehin nicht existierenden) Rückgabewert `varargout{1}` zugewiesen wird.

Ist also der letzte in das Texteingabefeld eingegebene Wert derjenige, der auch über das Kommandofenster ausgegeben werden soll, dann ist des Problems Lösung in einer Verlagerung der Funktion `uiresume` zu suchen. Erst dann wird ja die unterbrochene Öffnungsfunktion `07_Gui4Erweitert_OpeningFcn` fortgesetzt, in der das zusätzliche Element `mytext` von `handles` angelegt und vorbesetzt wurde.

Wo aber kann `uiresume` abgelegt werden?
Der letzte Eintrag liegt in dem Texteingabefeld sicherlich dann vor, wenn anschließend der GUI geschlossen wird. Bislang allerdings haben wir (anhand von Ausgaben des Kommandofensters) keine Reaktion bemerkt, wenn mittels Mausklick der GUI beendet wird. Eine solche Rückruf-Funktion können wir aber noch implementieren.

Erstellung einer „Close-Request-Function“
In der nächsten Version wird zusätzlich zu den bereits bestehenden Funktionen noch eine Funktion hinzugefügt, die immer dann aufgerufen wird, wenn der GUI geschlossen wird. Es handelt sich dabei um eine weitere Rückruf-Funktion, die unter dem Eintrag „Close-Request-Function“ in GUIDE zu finden ist.

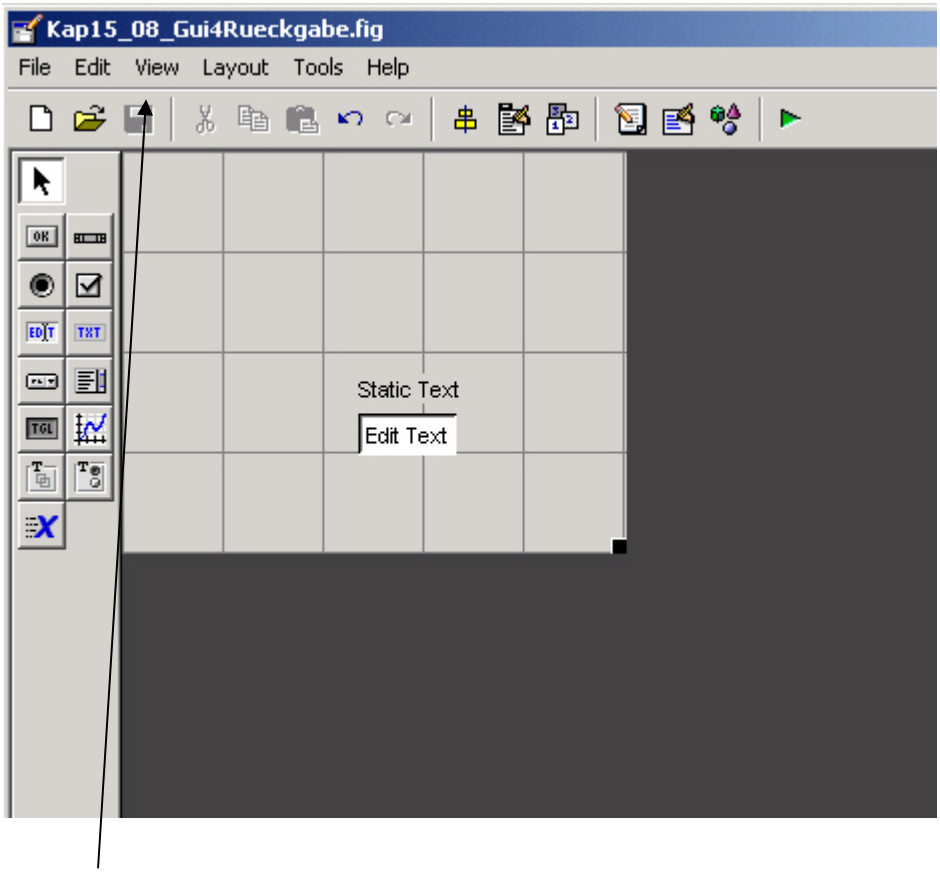
Legen wir also erneut eine Version des GUI mit einem Eingabetextfeld und einem Textfeld an, den wir unter „`Kap15_08_Gui4Rueckgabe`“ speichern. Dies soll dann (endlich) die komplette Version eines GUI’s werden, dessen Rückgabewert der letzte Eintrag im Texteingabefeld ist.

Wie bereits bekannt, werden nebst `gui_mainfcn` auch die editierbaren Funktionen

```
varargout  = Kap15_08_Gui4Rueckgabe(varargin),
Kap15_08_Gui4Rueckgabe_OpeningFcn(hObject, eventdata, handles, varargin),
varargout = Kap15_08_Gui4Rueckgabe_OutputFcn(hObject, eventdata, handles),
```

`edit1_Callback(hObject, eventdata, handles)` und
`edit1_CreateFcn(hObject, eventdata, handles)`
erzeugt.

Jetzt soll noch die Rückruf-Funktion hinzugefügt werden, die bei Schließen des GUI aufgerufen wird:



Menüeintrag "View"

Im Menüeintrag "View" wählt man aus unter „View Callbacks" den Untereintrag „`CloseRequestFcn`" aus. Im dazugehörigen Programm wird automatisch eine Funktion `figure1_CloseRequestFcn(hObject, eventdata, handles)` erzeugt, deren einziger Eintrag lautet:

```
delete(hObject);
```

Der ist noch etwas gefährlich – aber das demonstrieren wir gleich am lebenden Objekt.

In diese Funktion verlagern wir den Aufruf `uiresume`. Alle anderen Einträge lauten wie in der vorangegangenen Version des GUI's; das Programm hat somit folgendes Aussehen:

```
function varargout = Kap15_08_Gui4Rueckgabe(varargin)
% KAP15_08_GUI4RUECKGABE M-file for Kap15_08_Gui4Rueckgabe.fig
%      KAP15_08_GUI4RUECKGABE, by itself, creates a new
```

```
%      KAP15_08_GUI4RUECKGABE or raises the existing
%      singleton*.
%
%      H = KAP15_08_GUI4RUECKGABE returns the handle to a new
%      KAP15_08_GUI4RUECKGABE or the handle to
%      the existing singleton*.
%
%      KAP15_08_GUI4RUECKGABE('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in KAP15_08_GUI4RUECKGABE.M
%      with the given input arguments.
%
%      KAP15_08_GUI4RUECKGABE('Property','Value',...) creates a new
%      KAP15_08_GUI4RUECKGABE or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_08_Gui4Rueckgabe_OpeningFunction gets called.  An unrecognized
%      property name or invalid value makes property application stop.  All
%      inputs are passed to Kap15_08_Gui4Rueckgabe_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_08_Gui4Rueckgabe

% Last Modified by GUIDE v2.5 19-Jul-2007 18:17:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_08_Gui4Rueckgabe_OpeningFcn,
...
                   'gui_OutputFcn',  @Kap15_08_Gui4Rueckgabe_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_08_Gui4Rueckgabe is made visible.
function Kap15_08_Gui4Rueckgabe_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_08_Gui4Rueckgabe (see
%            VARARGIN)

% Choose default command line output for Kap15_08_Gui4Rueckgabe
handles.output = hObject;

handles.mytext = 'noch nix da';
% Abmessungen vergrößern
```

```
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
% Eingabetext zu Beginn
set(handles.edit1, 'String', 'Gib hier was ein');
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_08_Gui4Rueckgabe wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_08_Gui4Rueckgabe_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.mytext;


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double
txt = get(handles.edit1, 'String');
set(handles.text1, 'String', txt);
% Erweiterung der Struktur handles zur Aufnahme des Textes, um diesen
% an die Ausgabe-Funktion übergeben zu können
handles.mytext = txt;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
uiresume(handles.figure1);
delete(hObject);
```

Wie zu ersehen, ist nach wie vor `uiwait` in `Kap15_08_Gui4Rueckgabe_OpeningFcn` untergebracht, jedoch uiresume ausgelagert in die neue Rückruf-Funktion `figure1_CloseRequestFcn`.

Ruft man – wie bisher – im Kommandofenster auf

```
h = Kap15_08_Gui4Rueckgabe
```

erscheint auch brav der GUI.

Man tippt etwas in das Eingabefenster, schließt mit der Entertaste ab – brav wird der Text in das darüber befindliche Textfeld geschrieben, und dann will man den GUI beenden. Klicken mit der Maustaste auf das Kreuz zum Schließen bringt das Programm zum Absturz mit der Meldung:

```
??? Attempt to reference field of non-structure array.

Error in ==> Kap15_08_Gui4Rueckgabe>Kap15_08_Gui4Rueckgabe_OutputFcn at 79
varargout{1} = handles.mytext;

Error in ==> gui_mainfcn at 215
        [varargout{1:nargout}]       =       feval(gui_State.gui_OutputFcn,
gui_hFigure, [], gui_Handles);

Error in ==> Kap15_08_Gui4Rueckgabe at 40
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});

>>
```

Dies ist etwas verwirrend.

Die Meldung besagt, dass der Compiler `handles.mytext` nicht kennt. Andererseits wurde dieses Element bereits vorbesetzt, dann sogar mit einem neuen Wert belegt – und jetzt ist es plötzlich beim Aufruf von `Kap15_08_Gui4Rueckgabe_OutputFcn` plötzlich unbekannt.

Der Grund:

Kurz vor `Kap15_08_Gui4Rueckgabe_OutputFcn` wurde die Rückruf-Funktion `figure1_CloseRequestFcn` aufgerufen. Deren letzter Befehl lautet `delete(hObject)`. Damit ist nämlich auch `handles.mytext` gelöscht (und unbekannt), so dass `Kap15_08_Gui4Rueckgabe_OutputFcn` ins Leere greift.

Also kann der `delete`-Befehl an der Stelle nicht bleiben.

Nimmt man ihn allerdings weg, hat man eine Endlos-Schleife produziert. Der GUI lässt sich nicht mehr schließen; wenn man Glück hat, lässt sich das Programm noch mit der Tastenkombination Strg C beenden (sonst hilft nur noch der Affengriff Strg Alt Entf)

Wohin also mit dem `delete`-Befehl?

Da nach der Rückruf-Funktion `figure1_CloseRequestFcn` der Aufruf von `Kap15_08_Gui4Rueckgabe_OutputFcn` erfolgt, wird er nach dort verlagert.

Damit hat das korrigierte Programm die Fassung:

```
function varargout = Kap15_08_Gui4Rueckgabe(varargin)
% KAP15_08_GUI4RUECKGABE M-file for Kap15_08_Gui4Rueckgabe.fig
%      KAP15_08_GUI4RUECKGABE, by itself, creates a new
%      KAP15_08_GUI4RUECKGABE or raises the existing singleton*.
%
%      H = KAP15_08_GUI4RUECKGABE returns the handle to a new
%      KAP15_08_GUI4RUECKGABE or the handle to the existing singleton*.
%
%      KAP15_08_GUI4RUECKGABE('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in KAP15_08_GUI4RUECKGABE.M
%      with the given input arguments.
%
%      KAP15_08_GUI4RUECKGABE('Property','Value',...) creates a new
%      KAP15_08_GUI4RUECKGABE or raises the
%      existing singleton*.  Starting from the left, property value pairs
%      are applied to the GUI before Kap15_08_Gui4Rueckgabe_OpeningFunction
%      gets called.  An unrecognized property name or invalid value makes
%      property application stop.  All inputs are passed to
%      Kap15_08_Gui4Rueckgabe_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_08_Gui4Rueckgabe

% Last Modified by GUIDE v2.5 19-Jul-2007 18:17:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_08_Gui4Rueckgabe_OpeningFcn,
...
                   'gui_OutputFcn',  @Kap15_08_Gui4Rueckgabe_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_08_Gui4Rueckgabe is made visible.
function Kap15_08_Gui4Rueckgabe_OpeningFcn(hObject, eventdata, handles,
varargin)
```

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_08_Gui4Rueckgabe (see
%            VARARGIN)

% Choose default command line output for Kap15_08_Gui4Rueckgabe
handles.output = hObject;

handles.mytext = 'noch nix da';
% Abmessungen vergrößern
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
% Eingabetext zu Beginn
set(handles.edit1, 'String', 'Gib hier was ein');
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_08_Gui4Rueckgabe wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_08_Gui4Rueckgabe_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.mytext;
delete(handles.figure1);



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double
txt = get(handles.edit1, 'String');
set(handles.text1, 'String', txt);
% Erweiterung der Struktur handles zur Aufnahme des Textes, um diesen
% an die Ausgabe-Funktion übergeben zu können
handles.mytext = txt;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
uiresume(handles.figure1);
% delete(hObject);  wurde hier entfernt!
```

Jetzt funktioniert's endlich!

Erst wenn der GUI geschlossen und damit das Programm beendet wird, wird korrekt der ausgelesene Text des Eingabefeldes über `varargout` zurück- und im Kommandofenster ausgegeben.

Eine allerletzte Feinheit:

Während der GUI auf dem Bildschirm erscheint und auf eine Eingabe wartet, kann man immer noch in MATLAB in andere Programme wechseln, insbesondere im Kommandofenster von MATLAB weitere Befehle eingeben und damit Programme starten bzw. beenden. Für den Fall, dass es nur möglich sein soll, einen Text in das Texteingabefeld zu tippen und sonst alle anderen Aktionen von MATLAB zu sperren, kann dies durch ein zusätzliches Kommando geschehen. Es lautet:

```
set(hObject, 'Windows', 'modal')
```

Dies bauen wir in der Öffnungsfunktion vor `uiwait` ein.

Damit sind also in MATLAB bis auf Texteinabe jegliche anderen Aktionen blockiert. Die Blockade wird wieder aufgehoben, wenn der Befehl `uiresume` abgearbeitet, also wenn letztendlich der GUI geschlossen wird.

Nachfolgend endlich die endgültige Fassung, in der also mit Öffnen des GUI's nur noch eine Texteingabe und/oder Schließen desselben möglich ist:

```
function varargout = Kap15_08_Gui4Rueckgabe(varargin)
% KAP15_08_GUI4RUECKGABE M-file for Kap15_08_Gui4Rueckgabe.fig
%      KAP15_08_GUI4RUECKGABE, by itself, creates a new
%      KAP15_08_GUI4RUECKGABE or raises the existing singleton*.
%
%      H = KAP15_08_GUI4RUECKGABE returns the handle to a new
%      KAP15_08_GUI4RUECKGABE or the handle to the existing singleton*.
%
```

```
%      KAP15_08_GUI4RUECKGABE('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in KAP15_08_GUI4RUECKGABE.M
%      with the given input arguments.
%
%      KAP15_08_GUI4RUECKGABE('Property','Value',...) creates a new
%      KAP15_08_GUI4RUECKGABE or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_08_Gui4Rueckgabe_OpeningFunction gets called.  An unrecognized
%      property name or invalid value makes property application stop.  All
%      inputs are passed to Kap15_08_Gui4Rueckgabe_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_08_Gui4Rueckgabe

% Last Modified by GUIDE v2.5 19-Jul-2007 18:17:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_08_Gui4Rueckgabe_OpeningFcn,
...
                   'gui_OutputFcn',  @Kap15_08_Gui4Rueckgabe_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_08_Gui4Rueckgabe is made visible.
function Kap15_08_Gui4Rueckgabe_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_08_Gui4Rueckgabe (see
%            VARARGIN)

% Choose default command line output for Kap15_08_Gui4Rueckgabe
handles.output = hObject;

handles.mytext = 'noch nix da';
% Abmessungen vergrößern
set(handles.text1, 'Position', [24 8 20 1.154]);
set(handles.edit1, 'Position', [24 6 20 1.154]);
% Eingabetext zu Beginn
set(handles.edit1, 'String', 'Gib hier was ein');
% Update handles structure
guidata(hObject, handles);
```

```
%Blockieren des Bildschirms bis auf Handhabung mit dem GUI
set(hObject, 'Windows', 'modal');

% UIWAIT makes Kap15_08_Gui4Rueckgabe wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_08_Gui4Rueckgabe_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.mytext;
delete(handles.figure1);



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double
txt = get(handles.edit1, 'String');
set(handles.text1, 'String', txt);
% Erweiterung der Struktur handles zur Aufnahme des Textes, um diesen
% an die Ausgabe-Funktion übergeben zu können
handles.mytext = txt;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
uiresume(handles.figure1);
% delete(hObject);  wurde hier entfernt!
```

Falls – wie dies häufig bei vielen GUI's der Fall ist – noch ein separater Knopf zum Schließen des GUI's vorhanden ist, sollte der `uiresume`-Befehl in der zu diesem Knopf gehörigen untergebracht werden. Dann erfolgt nämlich der Aufruf der Ausgabe-Funktion, in der `varargout` die Ausgabedaten erhält, und anschließend der `delete`-Befehl zum Löschen des GUI's.

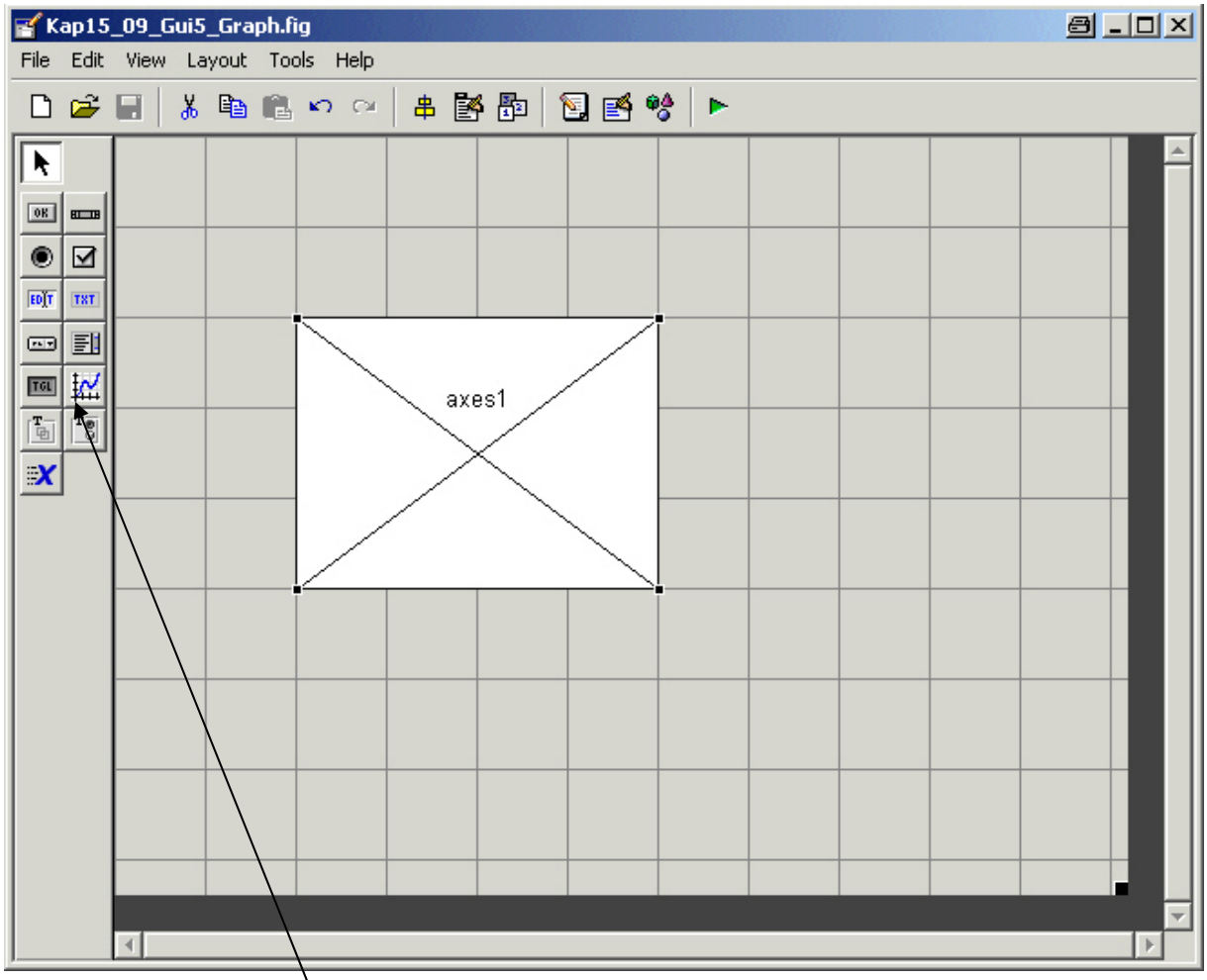
Graphik-Elemente in einem GUI

Als Beispiel der Anwendung eines Funktionsknopfes ließen wir einen Kreis zeichnen – dieser wurde als Graphik in einem separaten Fenster ausgegeben. Nun wollen wir ein solches Graphik-Element im GUI selbst placieren. Dazu wollen wir die dekorative Funktion

$$z = f(x,y) = \frac{\sin\left(\sqrt{x^2+y^2}\right)+\frac{1}{1000}}{\sqrt{x^2+y^2}+\frac{1}{1000}}$$

graphisch darstellen.

Wieder einmal erzeugen wir über GUIDE einen neuen GUI, in den wir ein sogenanntes „Axes"-Element einfügen. Dieses Element finden wir wie den Push-Button, das Texteingabe- und das Textfeld in der linken Auswahlpalette.



GUI-Element „Axes“ für graphische Darstellung

Den GUI speichern wir unter „`Kap15_09_Gui5_Graph.fig`“ ab.

Die dabei in „`Kap15_09_Gui5_Graph.m`“ erzeugten Funktionen ändern wir (zunächst) lediglich in der Öffnungsfunktion „`Kap15_09_Gui5_Graph_OpeningFcn`“ ab. Das Element hat den Namen „`axes1`“, also kann auf dessen Eigenschaften zugegriffen werden. Wir setzen nur (zum Testen) die Abmessungen und eben die Ausgabe eines dreidimensionalen Bildes.

Das vollständige Programm hat das Aussehen:

```
function varargout = Kap15_09_Gui5_Graph(varargin)
% KAP15_09_GUI5_GRAPH M-file for Kap15_09_Gui5_Graph.fig
%      KAP15_09_GUI5_GRAPH, by itself, creates a new KAP15_09_GUI5_GRAPH or
%      raises the existing singleton*.
%
%      H = KAP15_09_GUI5_GRAPH returns the handle to a new
%      KAP15_09_GUI5_GRAPH or the handle to the existing singleton*.
%
%      KAP15_09_GUI5_GRAPH('CALLBACK',hObject,eventData,handles,...) calls
%      the local function named CALLBACK in KAP15_09_GUI5_GRAPH.M with the
%      given input arguments.
%
%      KAP15_09_GUI5_GRAPH('Property','Value',...) creates a new
```

```
%      KAP15_09_GUI5_GRAPH or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_09_Gui5_Graph_OpeningFunction gets called.  An unrecognized
%      property name or invalid value makes property application stop.  All
%      inputs are passed to Kap15_09_Gui5_Graph_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_09_Gui5_Graph

% Last Modified by GUIDE v2.5 23-Jul-2007 12:42:12

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_09_Gui5_Graph_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_09_Gui5_Graph_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_09_Gui5_Graph is made visible.
function Kap15_09_Gui5_Graph_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_09_Gui5_Graph (see VARARGIN)

% Choose default command line output for Kap15_09_Gui5_Graph
handles.output = hObject;

% Graphik in aktuelle Achse zeichnen
x = -10 : 0.5 : 10;
y = x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.*xx + yy.*yy);
handles.z = (sin(r) + 0.001)./(r + 0.001);
surf(handles.z);

set(handles.axes1, 'Position', [10 10 70 20]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_09_Gui5_Graph wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = Kap15_09_Gui5_Graph_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Die neuen Teile in sind fett unterlegt.

Die Abmessungen werden wieder mit dem bekannten

```
set(handles.axes1, 'Position', [10 10 70 20]);
```

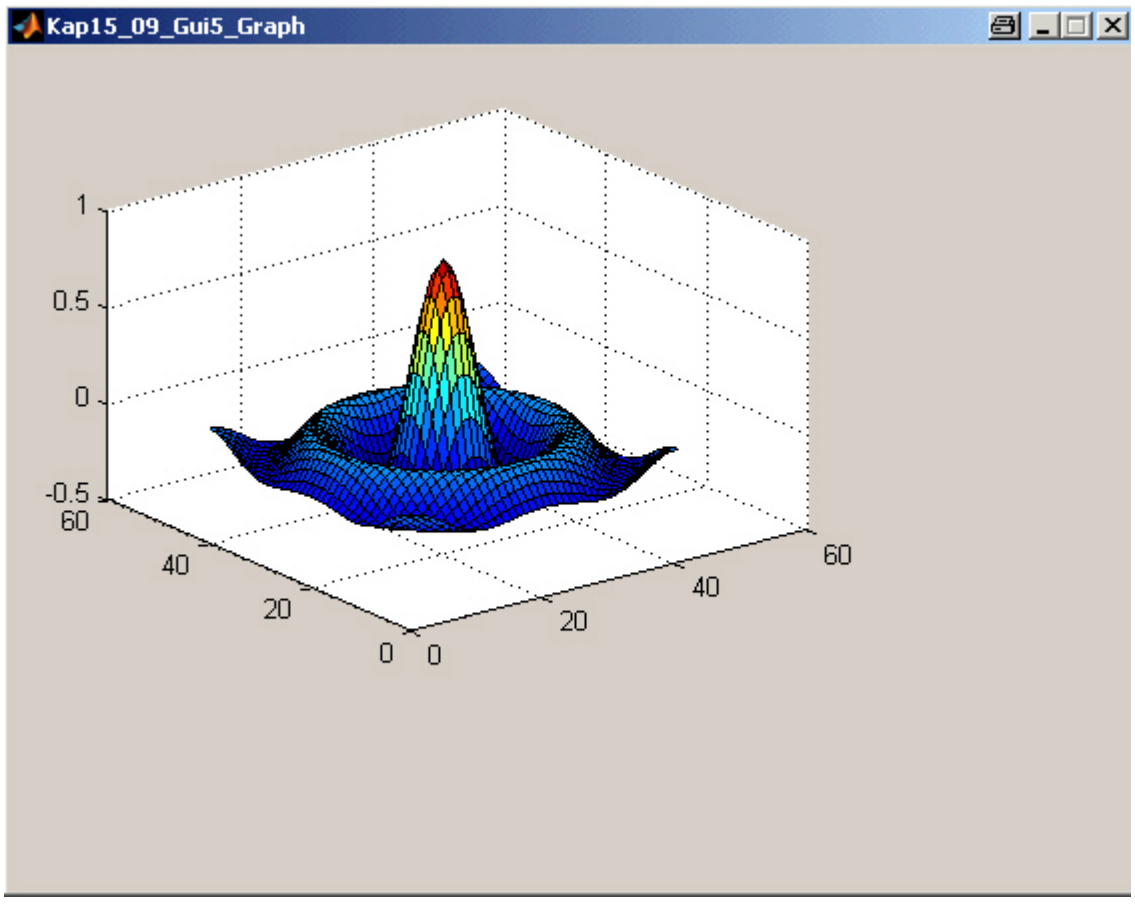
geregelt.

Der Graph wird zusätzlich in `handles.z` gespeichert. Der Sinn dieser Maßnahme wird in der nachfolgenden Erweiterung ersichtlich – für die aktuelle Darstellung sofort nach Darstellung des GUI's auf dem Bildschirm mit der graphischen Darstellung der Funktion f ist sie nicht erforderlich.

Jedenfalls erzeugt der Aufruf

```
>> Kap15_09_Gui5_Graph
>>
```

im Kommandofenster bereits das schöne Bild:



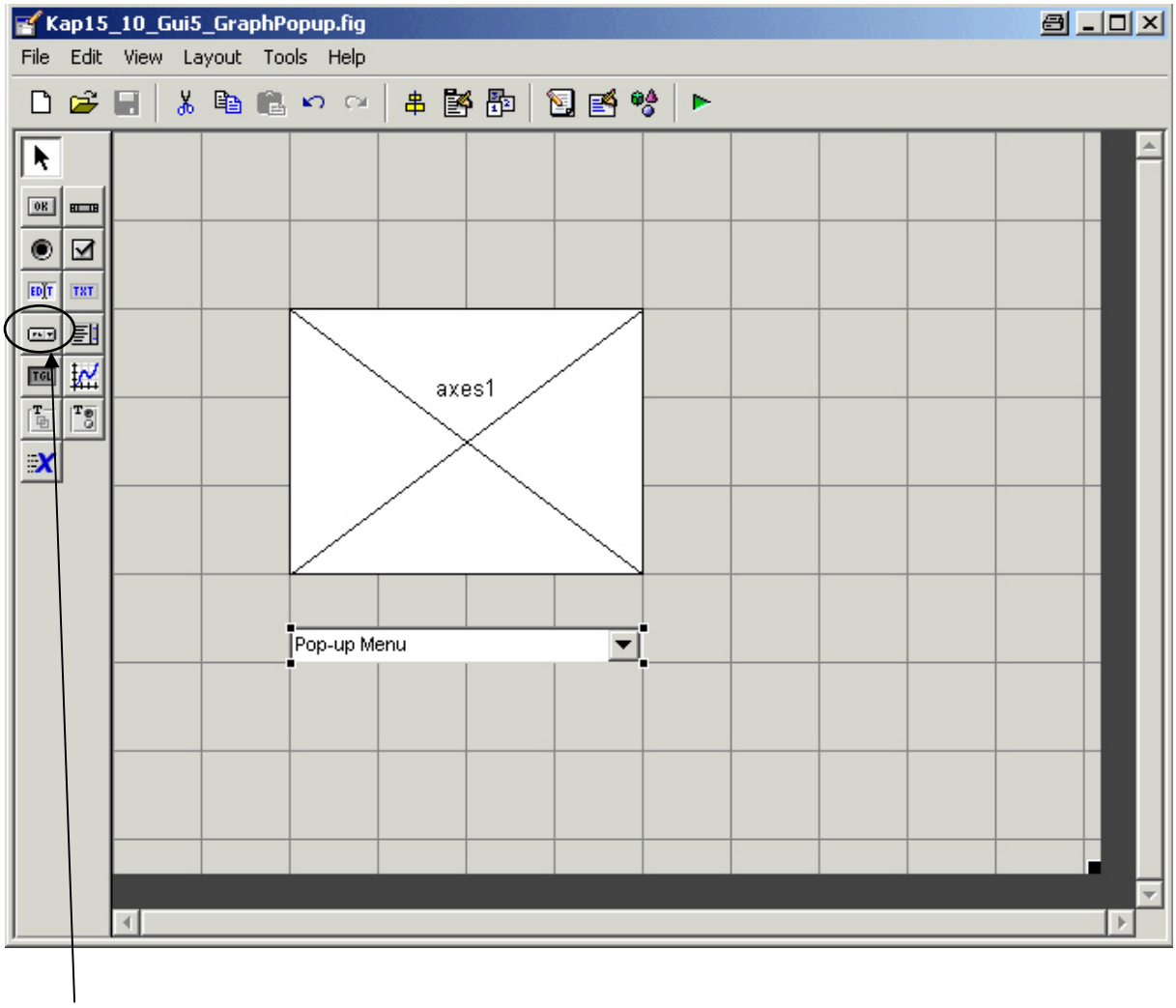
Somit wird also mit Aufruf eine 3D-Graphik gesetzt.

Dies ist nun doch etwas mager. Wir werden das Beispiel erweitern und dabei weitere GUI-Elemente vorstellen.

Pop-up-Menü

Die deutsche Bezeichnung dafür lautet „aufschlagbares Menüfeld“. Es handelt sich dabei um ein Feld innerhalb des GUI’s mit unterschiedlichen Einträgen, von denen der erste sichtbar ist und die anderen bei einem Mausklick auf das Feld angezeigt werden und ausgewählt werden können.

Wir werden ein solches in dem bereits erzeugten GUI des vorangegangenen Beispiels erzeugen (bzw. es wird ein neuer GUI angelegt, der zunächst erst einmal eine Kopie des vorangegangenen ist, aber unter „Kap15_10_Gui5_GraphPopup“ abgelegt ist). Zusätzlich zum Graphik-Fenster „`axes1`“ wird noch ein Pop-up_Menü „`popupmenu1`“ erzeugt. Man findet dieses ebenfalls in der linken Auswahlpalette.



Pop-up-Menü

Noch ist das Menü leer, d.h. es stehen keine Einträge zur Verfügung.
Um Einträge einzufügen, die dann per Mausklick ausgewählt werden können, gibt es zwei Möglichkeiten:

- Eintrag im Programm
- Eintrag über den Property-Inspector

Wir werden beide Möglichkeiten betrachten.

Eintrag im Programm

Die erste Möglichkeit besteht also darin, die auszuwählenden Einträge im dazugehörigen m-File in einer der Funktionen vorzunehmen, die mit Abspeichern des GUI's in „`Kap15_10_Gui5_GraphPopup.m`" erzeugt wurden. Als eine Möglichkeit steht die Create-Funktion „`popupmenu1_CreateFcn`" zur Verfügung.
Die Daten für das Pop-Up-Menü sind Cell-Objekte. Sie werden daher in folgender Form übergeben:

```
cell_var = {'Eintrag1', 'Eintrag2' , .... 'Eintrag_n'};
set(hObject, 'String', cell_var);
```

wobei `hObject` – wenn es in der Create-Funktion gesetzt ist – das Objekt zu dem Handle auf das Pop-up-Menü ist
(oder auch `handles.popmenu1`)

In unserem Fall wollen wir die Darstellungsart auswählen:

```
mesh
surf
```

dies sind die beiden Einträge – und werden entsprechend in der Create-Funktion gesetzt

```
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% Pop-up-Menü belegen
cell_var = {'mesh', 'surf'};
set(hObject, 'String', cell_var);
```

In der dazugehörigen Rückruf-Funktion „`popupmenu1_Callback`" wird dann auf die ausgewählten Einträge reagiert.

Dort wird nach dem angewählten Eintrag abgefragt. Jeder der n Einträge ist als Cell-Objekt abgelegt, weswegen der Inhalt über die Form

```
str = get(handles.popupmenu1, 'String');
if str{Eintragsnummer} = Eintrag1 ….
```

überprüft werden kann.

Das komplette Programm hat das Aussehen:

```
function varargout = Kap15_10_Gui5_GraphPopup(varargin)
% KAP15_10_GUI5_GRAPHPOPUP M-file for Kap15_10_Gui5_GraphPopup.fig
%      KAP15_10_GUI5_GRAPHPOPUP, by itself, creates a new
%      KAP15_10_GUI5_GRAPHPOPUP or raises the existing
%      singleton*.
%
%      H = KAP15_10_GUI5_GRAPHPOPUP returns the handle to a new
%      KAP15_10_GUI5_GRAPHPOPUP or the handle to the existing singleton*.
%
%      KAP15_10_GUI5_GRAPHPOPUP('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in
%      KAP15_10_GUI5_GRAPHPOPUP.M with the given input arguments.
%
%      KAP15_10_GUI5_GRAPHPOPUP('Property','Value',...) creates a new
%      KAP15_10_GUI5_GRAPHPOPUP or raises the existing singleton*.
%      Starting from the left, property value pairs are applied to the GUI
%      before Kap15_10_Gui5_GraphPopup_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to
%      Kap15_10_Gui5_GraphPopup_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% Kap15_10_Gui5_GraphPopup

% Last Modified by GUIDE v2.5 23-Jul-2007 13:16:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_10_Gui5_GraphPopup_OpeningFcn,
...
                   'gui_OutputFcn',  @Kap15_10_Gui5_GraphPopup_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

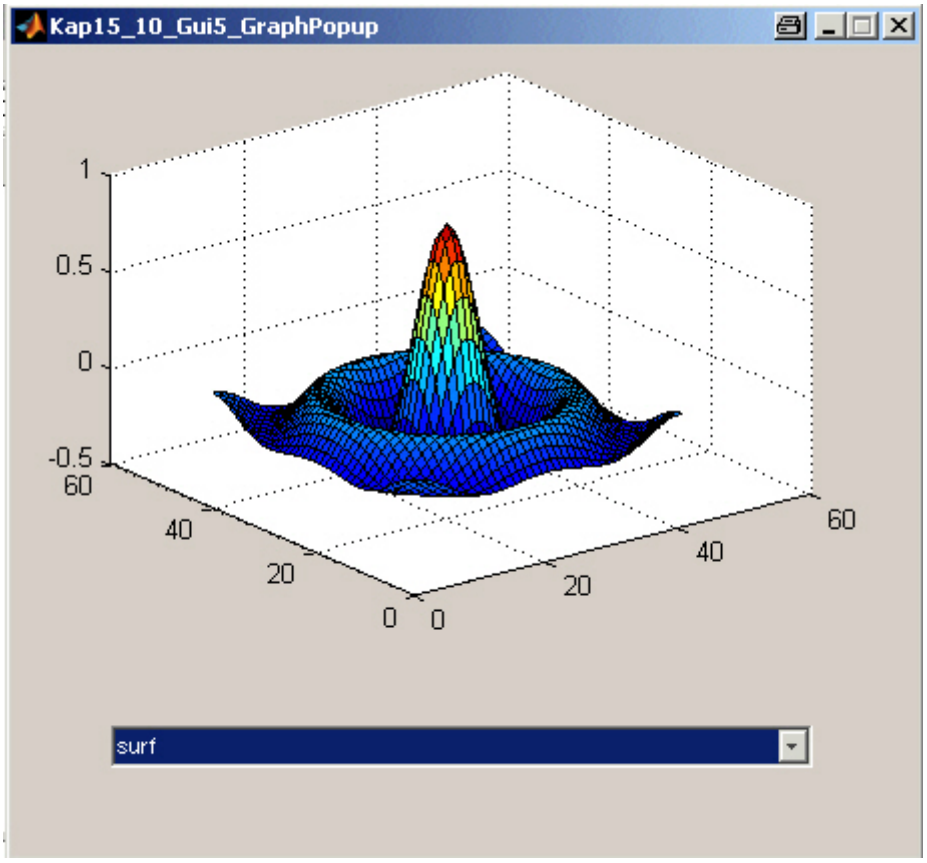
```
% --- Executes just before Kap15_10_Gui5_GraphPopup is made visible.
function Kap15_10_Gui5_GraphPopup_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_10_Gui5_GraphPopup (see
%            VARARGIN)

% Choose default command line output for Kap15_10_Gui5_GraphPopup
handles.output = hObject;

% Graphik in aktuelle Achse zeichnen
x = -10 : 0.5 : 10;
y = x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.*xx + yy.*yy);
handles.z = (sin(r) + 0.001)./(r + 0.001);
surf(handles.z);

set(handles.axes1, 'Position', [10 10 70 20]);


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_10_Gui5_GraphPopup wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_10_Gui5_GraphPopup_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
%        cell array
%        contents{get(hObject,'Value')} returns selected item from
%        popupmenu1

% Nummer des ausgewählten Menü-Eintrags
val = get(handles.popupmenu1, 'Value');
% Liste der Einträge im Pop-up-Menü
str = get(handles.popupmenu1, 'String');

% Text zur ausgewählten  Nummer des Eintrags
switch (str{val})
    case 'surf'
```

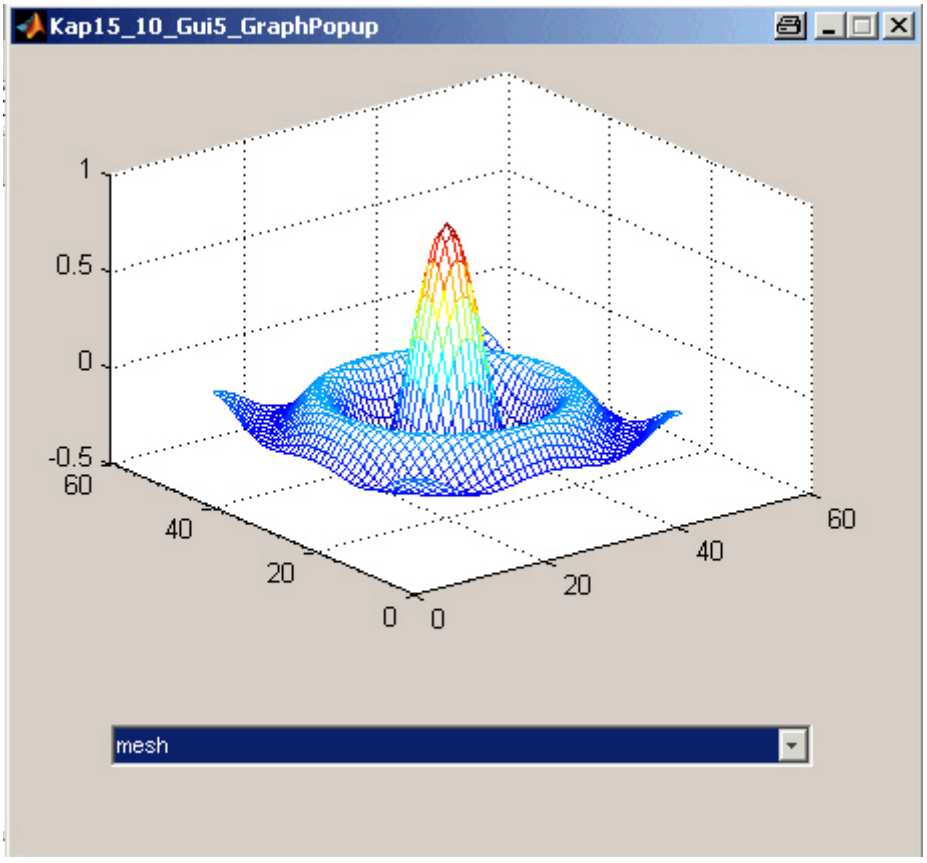
```
        surf(handles.z);
    case 'mesh'
        mesh(handles.z);
end

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Pop-up-Menü belegen
cell_var = {'mesh', 'surf'};
set(hObject, 'String', cell_var);
set(hObject, 'Position', [10 -15 70 20]);
```

Die hinzugefügten Teile sind wieder fett unterlegt.

Jetzt wird auch verständlich, warum in der Öffnungsfunktion „`Kap15_10_Gui5_GraphPopup_OpeningFcn`“ die Werte des Graphens zwischen gespeichert wurden – auf diese greift dann in `popupmenu1_Callback` das eigentliche Zeichenprogramm `mesh` bzw. `surf` in Abhängigkeit des ausgewählten Eintrags zu.

Somit lässt sich also nach Start des Programms jetzt im Pop-up-Menü zwischen den Darstellungen „`mesh`“ und „`surf`“ umschalten:


Kap15_10_Gui5_GraphPopup
1
0.5
0
-0.5
60
40
20
0
0
20
40
60
surf

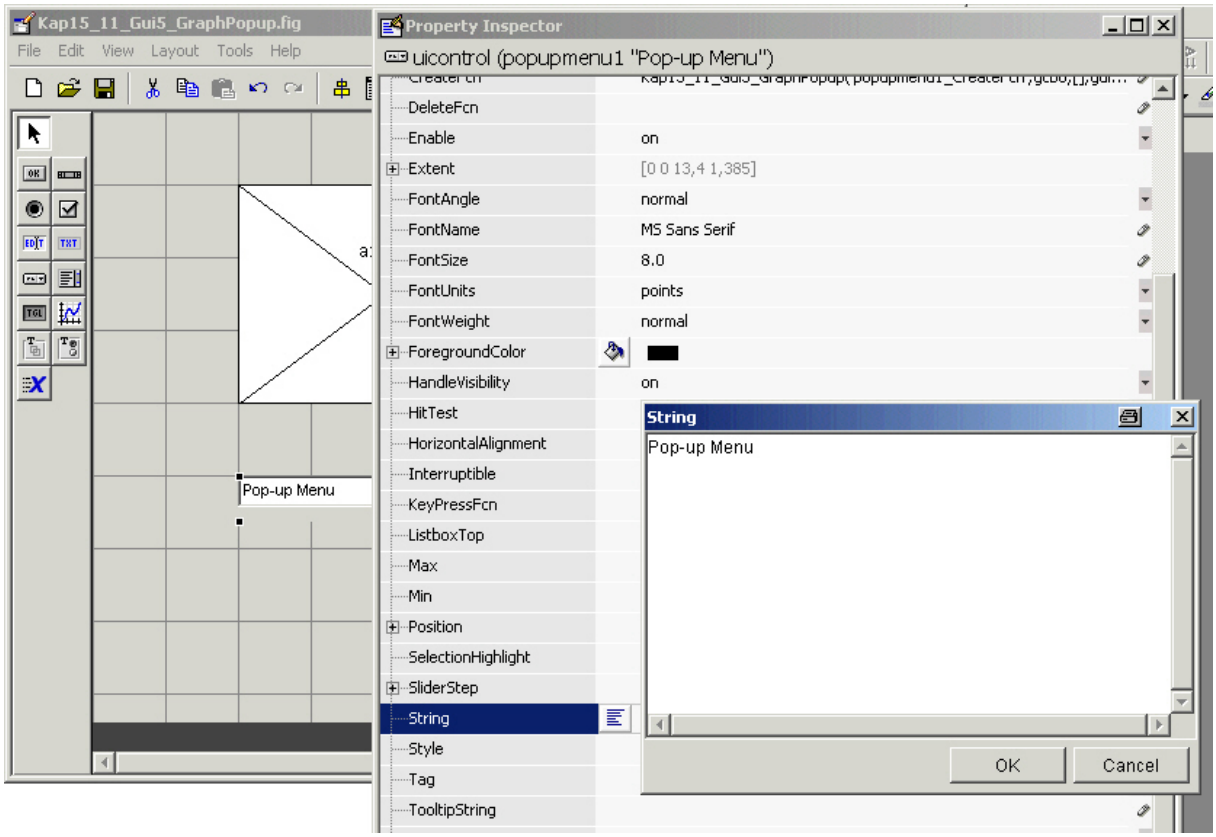

Kap15_10_Gui5_GraphPopup
1
0.5
0
-0.5
60
40
20
0
0
20
40
60
mesh

Eintrag über Property-Inspector

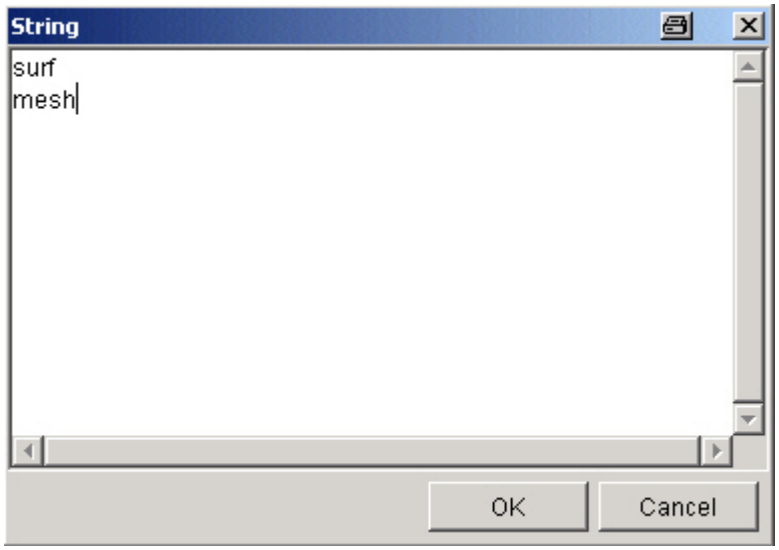
Die andere Möglichkeit, die auszuwählenden Texte im Pop-up-Menü einzutragen, besteht darin, in GUIDE den Property-Inspector aufzurufen.

Diese Möglichkeit wollen wir noch kurz anmerken (hier: neuen GUI mit Graphik-Element und Pop-up-Menü erzeugen und unter „Kap15_11_Gui5_GraphPopup.fig“ ablegen).

Wenn also in GUIDE das Pop-up-Menü im GUI placiert worden ist, erhält man, wenn dieses mit rechtem Mausklick angeklickt wird, ein Kontextmenü, indem der Property-Inspector ausgewählt werden kann. Dieser enthält sämtliche Eigenschaften dieses GUI-Elements. Insbesondere befindet sich beim Eintrag „String“; an dieser Stelle können die Einträge für das Menü editiert werden. Man klickt dazu mutig den Knopf mit den blauen Streifen an und erhält zur Belohnung ein weiteres Fenster geöffnet, in dem nun (endlich) die gewünschten Einträge vorgenommen werden können:



Für unsere Zwecke sieht ein Eintrag dann so aus:



Man drückt den „OK“ – Knopf, schließt den Property-Inspector und GUIDE – und das war’s dann auch schon.

Das modifizierte Programm hat das Aussehen:

```
function varargout = Kap15_11_Gui5_GraphPopup(varargin)
% KAP15_11_GUI5_GRAPHPOPUP M-file for Kap15_11_Gui5_GraphPopup.fig
%      KAP15_11_GUI5_GRAPHPOPUP, by itself, creates a new
%      KAP15_11_GUI5_GRAPHPOPUP or raises the existing singleton*.
%
%      H = KAP15_11_GUI5_GRAPHPOPUP returns the handle to a new
%      KAP15_11_GUI5_GRAPHPOPUP or the handle to the existing singleton*.
%
%      KAP15_11_GUI5_GRAPHPOPUP('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in
%      KAP15_11_GUI5_GRAPHPOPUP.M with the given input arguments.
%
%      KAP15_11_GUI5_GRAPHPOPUP('Property','Value',...) creates a new
%      KAP15_11_GUI5_GRAPHPOPUP or raises the existing singleton*.
%      Starting from the left, property value pairs are
%      applied to the GUI before Kap15_11_Gui5_GraphPopup_OpeningFunction
%      gets called.  An unrecognized property name or invalid value makes
%      property application stop.  All inputs are passed to
%      Kap15_11_Gui5_GraphPopup_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% Kap15_11_Gui5_GraphPopup

% Last Modified by GUIDE v2.5 23-Jul-2007 14:21:43

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_11_Gui5_GraphPopup_OpeningFcn,
...
```

```
                   'gui_OutputFcn',  @Kap15_11_Gui5_GraphPopup_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_11_Gui5_GraphPopup is made visible.
function Kap15_11_Gui5_GraphPopup_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_11_Gui5_GraphPopup (see
%            VARARGIN)

% Choose default command line output for Kap15_11_Gui5_GraphPopup
handles.output = hObject;

% Graphik in aktuelle Achse zeichnen
x = -10 : 0.5 : 10;
y = x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.*xx + yy.*yy);
handles.z = (sin(r) + 0.001)./(r + 0.001);
surf(handles.z);

set(handles.axes1, 'Position', [10 10 70 20]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_11_Gui5_GraphPopup wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_11_Gui5_GraphPopup_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
%        cell array
%        contents{get(hObject,'Value')} returns selected item from
%        popupmenu1

% Nummer des ausgewählten Menü-Eintrags
val = get(handles.popupmenu1, 'Value');
% Liste der Einträge im Pop-up-Menü
str = get(handles.popupmenu1, 'String');

% Text zur ausgewählten  Nummer des Eintrags
switch (str{val})
    case 'surf'
        surf(handles.z);
    case 'mesh'
        mesh(handles.z);
end

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'Position', [10 -15 70 20]);
```

Natürlich fehlen jetzt in der Create-Funktion `popupmenu1_CreateFcn` die Einträge für das Pop-up-Menü – dies ist im Property-Inspector enthalten. Lediglich die Abmessungen für das Menü wurden aus dem vorangegangenen Programm übernommen. Die anderen Anweisungen sind aus dem vorangegangenen Programm übernommen.

Nachfolgend einige Variationen des Programms durch Gebrauch anderer GUI-Elemente, die in der Auswahlpalette von GUIDE aufgeführt sind.

Listbox

Auf Deutsch kann dies mit „Auswahlliste“ übersetzt werden. Die Funktionalität ähnelt der eines Pop-up-Menüs. Während dieses allerdings nur den obersten Eintrag anzeigt und die restlichen erst bei Anklicken, werden in einer Listbox alle Elemente aufgeführt – sofern genügend Platz vorhanden ist.

Daher nachfolgend die Variation mit einer Listbox:

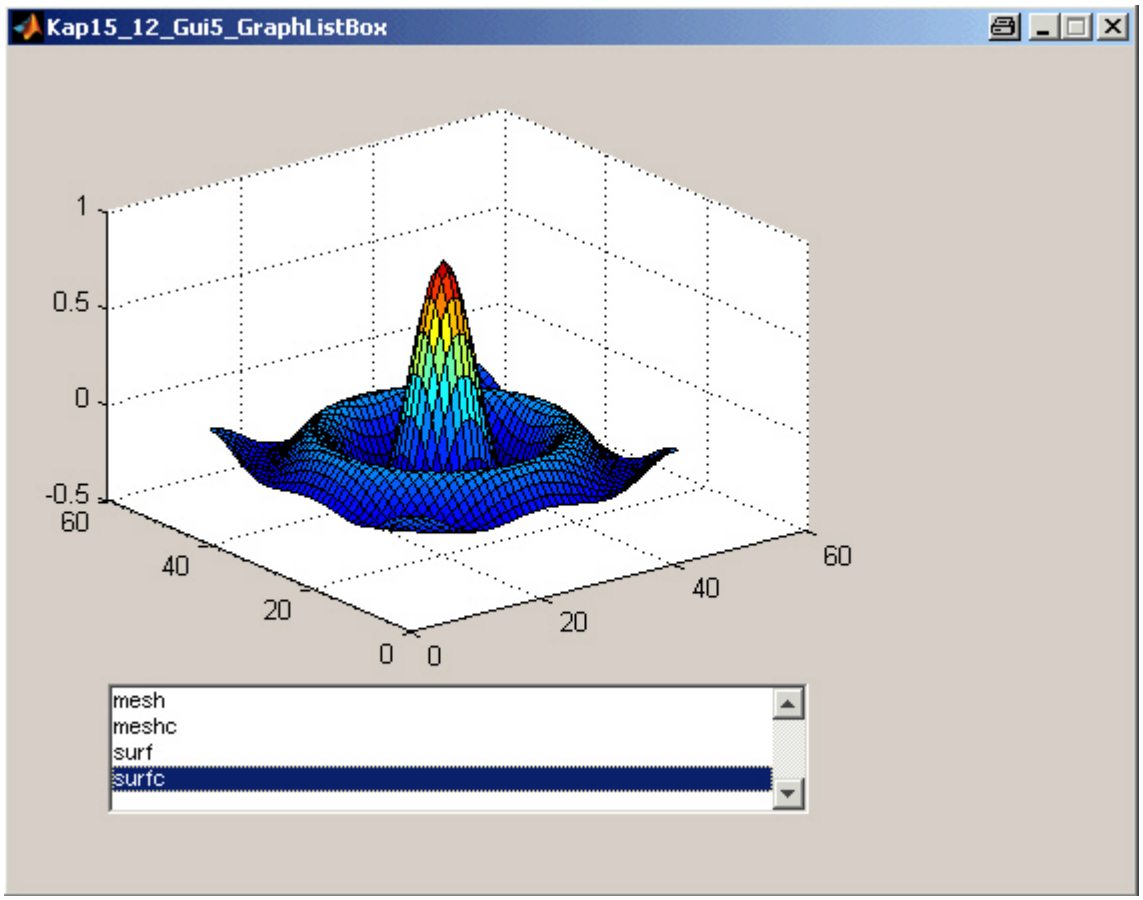
```
function varargout = Kap15_12_Gui5_GraphListBox(varargin)
% KAP15_12_GUI5_GRAPHLISTBOX M-file for Kap15_12_Gui5_GraphListBox.fig
%      KAP15_12_GUI5_GRAPHLISTBOX, by itself, creates a new
%      KAP15_12_GUI5_GRAPHLISTBOX or raises the existing singleton*.
%
%      H = KAP15_12_GUI5_GRAPHLISTBOX returns the handle to a new
%      KAP15_12_GUI5_GRAPHLISTBOX or the handle to the existing singleton*.
%
%      KAP15_12_GUI5_GRAPHLISTBOX('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in
%      KAP15_12_GUI5_GRAPHLISTBOX.M with the given input arguments.
%
%      KAP15_12_GUI5_GRAPHLISTBOX('Property','Value',...) creates a new
%      KAP15_12_GUI5_GRAPHLISTBOX or raises the existing singleton*.
%      Starting from the left, property value pairs are applied to the GUI
%      before Kap15_12_Gui5_GraphListBox_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to
%      Kap15_12_Gui5_GraphListBox_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% Kap15_12_Gui5_GraphListBox

% Last Modified by GUIDE v2.5 23-Jul-2007 17:19:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@Kap15_12_Gui5_GraphListBox_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_12_Gui5_GraphListBox_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_12_Gui5_GraphListBox is made visible.
function Kap15_12_Gui5_GraphListBox_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_12_Gui5_GraphListBox (see
            VARARGIN)
```

```
% Choose default command line output for Kap15_12_Gui5_GraphListBox
handles.output = hObject;

x = -10 : 0.5 : 10;
y = x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.*xx + yy.*yy);
handles.z = (sin(r) + 0.001)./(r + 0.001);
surf(handles.z);

set(handles.axes1, 'Position', [10 10 70 20]);


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_12_Gui5_GraphListBox wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_12_Gui5_GraphListBox_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox1 contents as cell
%        array
%        contents{get(hObject,'Value')} returns selected item from listbox1

% Nummer des ausgewählten Menü-Eintrags
val = get(handles.listbox1, 'Value');
% Liste der Einträge im Pop-up-Menü
str = get(handles.listbox1, 'String');

% Text zur ausgewählten  Nummer des Eintrags
switch (str{val})
    case 'surf'
        surf(handles.z);
    case 'surfc'
        surfc(handles.z);
    case 'mesh'
        mesh(handles.z);
    case 'meshc'
        meshc(handles.z);
end

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
```

```
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Pop-up-Menü belegen
cell_var = {'mesh', 'meshc', 'surf', 'surfc'};
set(hObject, 'String', cell_var);

set(hObject, 'Position', [10 3 70 5]);
```

Man sieht sehr schnell, daß dieses Programm im Prinzip bis auf die Namensgebung eine getreue Kopie des vorangegangenen ist – außer daß aus Spaß an der Freud statt nur zwei es immerhin vier Auswahlmöglichkeiten gibt.

Die Einträge für die Listbox können ebenso über den Property-Inspector vorgenommen werden, indem man die Einträge für „String“ modifiziert.
Eine Ausgabe hat folgendes Aussehen:



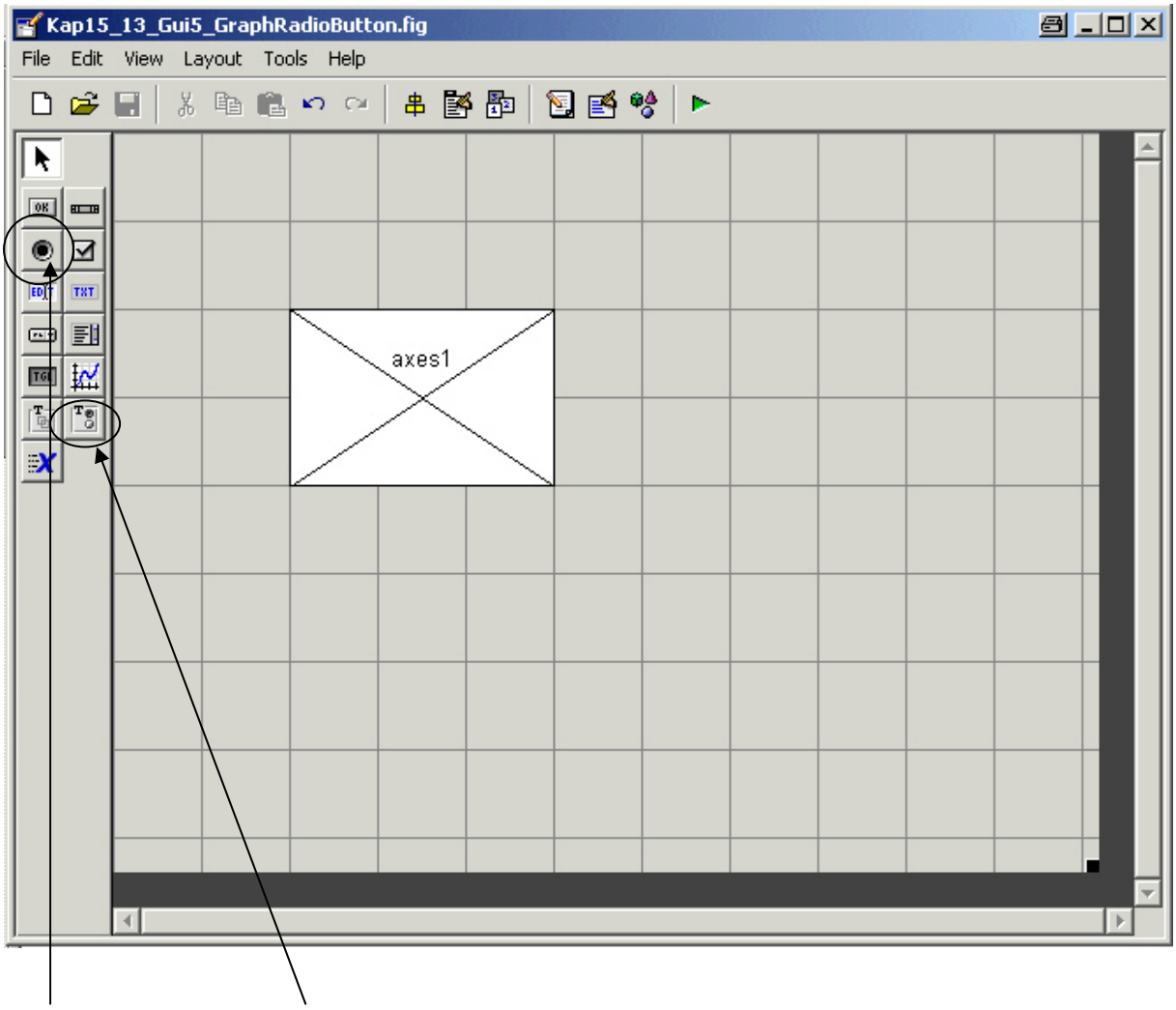
Radio-Buttons (und Gruppierung)

Radio-Buttons – oder auf Deutsch „Optionalschaltflächen“ - bestehen aus mehreren Schaltflächen in Form von Knöpfen, von denen nur eine angewählt werden kann; die anderen sind alle inaktiv. Nur die aktive Schaltfläche kann eine Aktion auslösen. Zwei gleichzeitig aktive Schaltflächen gibt es nicht (daher auch der Name „Radio“ – wie bei einem Radio der Senderknopf unter vielen Sendern nur genau einen auswählen kann, so hat man diverse Optionen, von denen aber nur eine gewählt werden kann).

Die Auswahlmöglichkeiten werden wieder die Darstellungsmöglichkeiten des Graphen sein, nämlich „mesh“, „meshc“, „surf“, „surfc“.

Damit MATLAB „weiß“, dass eine bestimmte Anzahl von Schaltflächen sich gegenseitig ausschließen, müssen diese zu einer Gruppe zusammengefasst sein (immerhin ist ja noch eine weitere Gruppe von Optionsschaltflächen denkbar, die unabhängig von der ersten agiert). Dies geschieht mit einem sogenannten „Button Group“.

Die Erzeugung geschieht wieder wie bekannt in GUIDE.
Wieder wird ein leerer GUI erzeugt, in dem wir für die Aufnahme der Graphik das Element „Axis“ anlegen. Den GUI speichern wir unter „`Kap15_13_Gui5_GraphRadioButton.fig`“ ab.



Radio-Button Button Group

Wichtig ist jetzt die Reihenfolge!

Zuerst wird für die Gruppierung „Button Group“ ausgewählt und im GUI placiert.

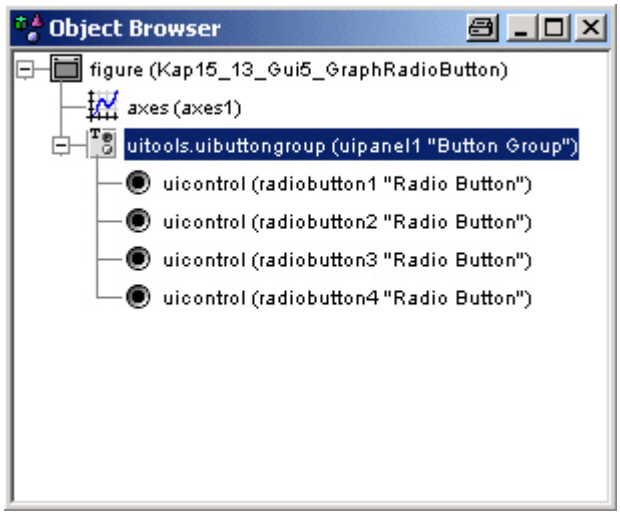
Danach werden (insgesamt vier Mal) die Radio-Buttons ausgewählt und innerhalb der Button Group positioniert – auf diese Weise wird MATLAB mitgeteilt, dass diese Schaltflächen zu einer Gruppe zusammengefasst sind[2]. Der zuerst eingefügte Knopf wird automatisch auf aktiv gesetzt.

Der GUI hat damit folgendes Aussehen:

Daß die Schaltflächen zu einer Gruppe zusammengefasst sind, lässt sich in der sogenannten Objekt-Hierarchie ersehen:

[2] man kann die Radio-Buttons auch ohne vorherige Gruppierung einfügen; dann aber funktionieren sie unabhängig voneinander, d.h. es kann mehr als eine Schaltfläche gesetzt werden.

Rechter Mausklick auf „Button Group“ oder einer der Knöpfe öffnet das Kontextmenü, über das wir sonst immer den Property Inspector öffneten. Diesmal wählen wir den Eintrag „Object Browser“ aus; er stellt die bereits verwendeten Elemente in dem GUI graphisch dar:



Die verwendeten Elemente sind eigentlich Objekte und können damit eine Hierarchie haben. Daß jetzt die Optionalschaltflächen Radiobutton1 bis 4 zu einer Gruppe zusammengefasst sind, ist dadurch realisiert, dass die Button Group als Eltern- und die Knöpfe als davon abhängige Kinder-Objekte implementiert sind.

Betrachtet man die dazugehörige Datei „`Kap15_13_Gui5_GraphRadioButton.m`“, wird man erstaunt feststellen, dass keine Rückruf-Funktion erstellt worden ist – weder für die Gruppierung „`uipanel1`“ noch für die einzelnen Knöpfe „`radiobutton1`“ bis „`..4`“. Dies muß dem System noch separat mitgeteilt werden.

Was geschehen soll, wenn ein Knopf betätigt wird, übernimmt eine sogenannte Selektier-Funktion, die der Gruppierung zugeordnet ist. Diese muß noch hinzugefügt werden.

Dazu klickt man in GUIDE wieder mit der rechten Maustaste das Gruppen-Element mit der Überschrift „Button Group“ an und wählt im Kontextmenü den Eintrag „View Callbacks“ aus (oder im Menü-Eintrag „View“ -> „View Callbacks“). Im Untermenü wählt man „SelectionChangeFcn“ aus.

In „`Kap15_13_Gui5_GraphRadioButton.m`“ wird eine neue Funktion „`uipanel1_SelectionChangeFcn(hObject, eventdata, handles)`“ hinzugefügt. Diese wird nach folgendem Schema ausgefüllt:

```
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
```

```
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch get(hObject,'Tag')   % Get Tag of selected object
    case 'radiobutton1'
        % Code for when radiobutton1 is selected.
        surf(handles.z);
    case 'radiobutton2'
        % Code for when radiobutton2 is selected.
        surfc(handles.z);
    case 'radiobutton3'
        % Code for when togglebutton1 is selected.
        mesh(handles.z);
    case 'radiobutton4'
        % Code for when togglebutton2 is selected.
        meshc(handles.z);
    otherwise
        % Code for when there is no match.
end
```

Dann werden noch in der Öffnungsfunktion die Abmessungen gesetzt, das Element `handles.z` erzeugt, so dass das vollständige Programm das Aussehen hat:

```
function varargout = Kap15_13_Gui5_GraphRadioButton(varargin)
% KAP15_13_GUI5_GRAPHRADIOBUTTON M-file for
Kap15_13_Gui5_GraphRadioButton.fig
%      KAP15_13_GUI5_GRAPHRADIOBUTTON, by itself, creates a new
%      KAP15_13_GUI5_GRAPHRADIOBUTTON or raises the existing singleton*.
%
%      H = KAP15_13_GUI5_GRAPHRADIOBUTTON returns the handle to a new
%      KAP15_13_GUI5_GRAPHRADIOBUTTON or the handle to  the existing
%      singleton*.
%
%
KAP15_13_GUI5_GRAPHRADIOBUTTON('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in
%      KAP15_13_GUI5_GRAPHRADIOBUTTON.M with the given input arguments.
%
%      KAP15_13_GUI5_GRAPHRADIOBUTTON('Property','Value',...) creates a new
%      KAP15_13_GUI5_GRAPHRADIOBUTTON or raises the existing singleton*.
%      Starting from the left, property value pairs are applied to the GUI
%      before Kap15_13_Gui5_GraphRadioButton_OpeningFunction gets called.
%      An unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to
%      Kap15_13_Gui5_GraphRadioButton_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% Kap15_13_Gui5_GraphRadioButton

% Last Modified by GUIDE v2.5 24-Jul-2007 11:15:06

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn',
@Kap15_13_Gui5_GraphRadioButton_OpeningFcn, ...
                   'gui_OutputFcn',
@Kap15_13_Gui5_GraphRadioButton_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_13_Gui5_GraphRadioButton is made visible.
function Kap15_13_Gui5_GraphRadioButton_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_13_Gui5_GraphRadioButton (see
%            VARARGIN)

% Choose default command line output for Kap15_13_Gui5_GraphRadioButton
handles.output = hObject;

x = -10 : 0.5 : 10;
y = x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.*xx + yy.*yy);
handles.z = (sin(r) + 0.001)./(r + 0.001);
surf(handles.z);

set(handles.axes1, 'Position', [10 10 70 20]);

set(handles.uipanel1, 'Title', 'Auswahl');
set(handles.uipanel1, 'Position', [83 2 20 10]);

set(handles.radiobutton1, 'String', 'surf');
set(handles.radiobutton1, 'Position', [1 7  18 2]);
set(handles.radiobutton2, 'String', 'surfc');
set(handles.radiobutton2, 'Position', [1 5 18 2]);
set(handles.radiobutton3, 'String', 'mesh');
set(handles.radiobutton3, 'Position', [1 3 18 2]);
set(handles.radiobutton4, 'String', 'mesh');
set(handles.radiobutton4, 'Position', [1 1 18 2]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_13_Gui5_GraphRadioButton wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_13_Gui5_GraphRadioButton_OutputFcn(hObject,
eventdata, handles)
```

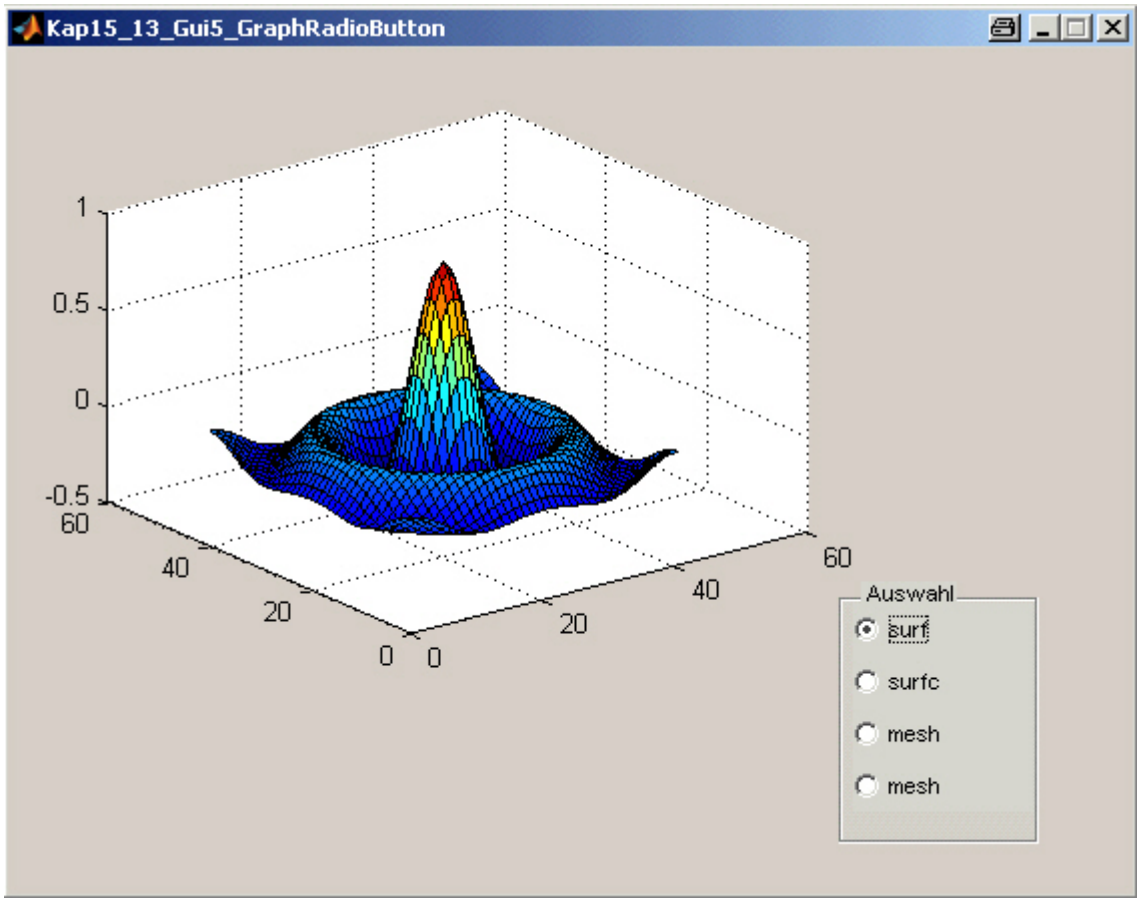
```
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(hObject,'Tag')   % Get Tag of selected object
    case 'radiobutton1'
        % Code for when radiobutton1 is selected.
        surf(handles.z);
    case 'radiobutton2'
        % Code for when radiobutton2 is selected.
        surfc(handles.z);
    case 'radiobutton3'
        % Code for when togglebutton1 is selected.
        mesh(handles.z);
    case 'radiobutton4'
        % Code for when togglebutton2 is selected.
        meshc(handles.z);
    otherwise
        % Code for when there is no match.
        warning('Fall bei Aufruf Radio Buttons vergessen!');
end
```

Bei den Abmessungen der Radio-Buttons ist noch zu sagen, dass diese, da sie Kinder-Objekte des Gruppierungs-Objektes „`uipanel1`“ sind, innerhalb der Abmessungen von „`uipanel1`“ gesetzt werden.

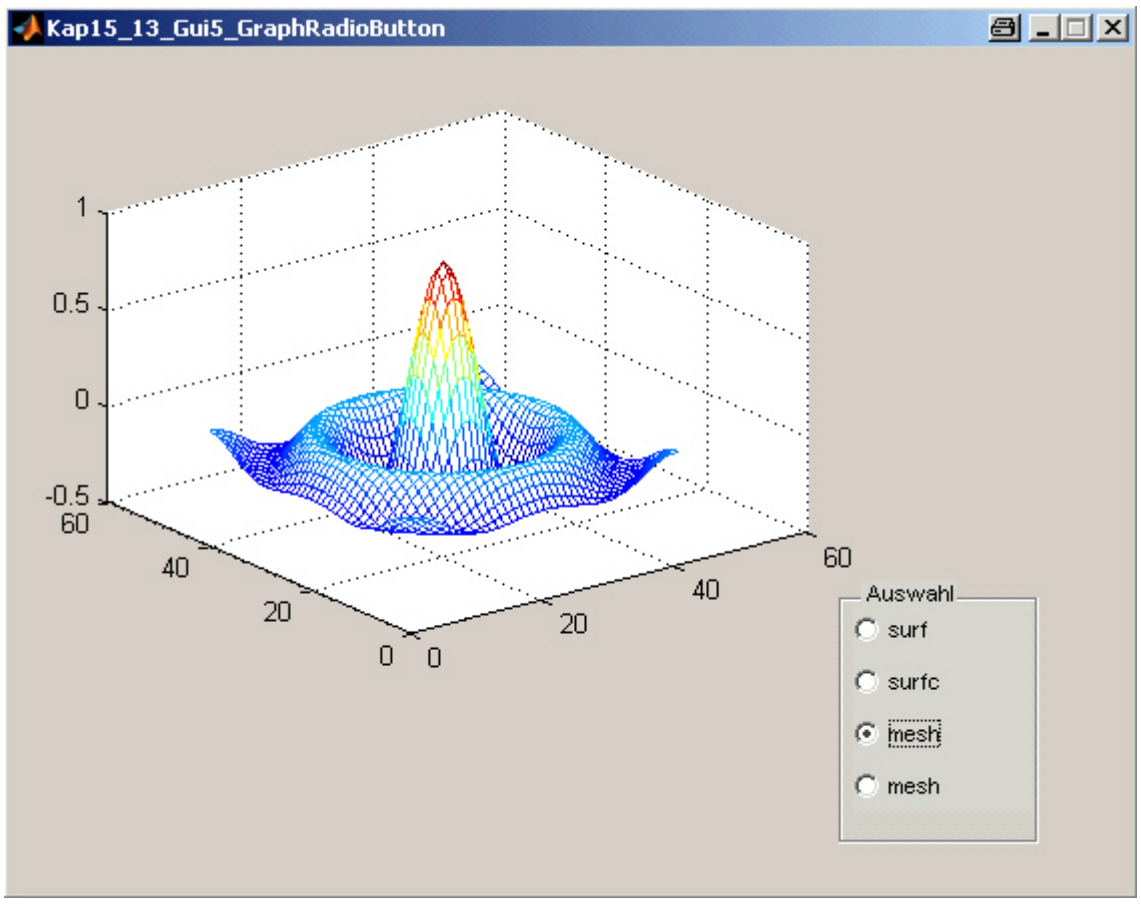
Wie bereits erwähnt, wird der erste Knopf standardmäßig bei Beginn aktiviert, so dass nach Aufruf

```
>> Kap15_13_Gui5_GraphRadioButton
>>
```

im Kommandofenster folgendes Fenster auf dem Bildschirm ausgegeben wird:



Klickt man danach „`mesh`“ an, wird das Bild wie folgt modifiziert:



Weitere GUI-Elemente (genaugenommen Objekte), die der Auswahlpalette in GUIDE entnommen werden können, sind:

Toggle Button	Ein- und Ausschaltknopf
Check Box	Ankreuz-Option oder auch Kontrollfeld genannt
Slider	Schieberegler zur Anzeige von Daten innerhalb eines Intervalls
Panel	zum Gruppieren von GUI-Elementen mit Titel, Rand etc.
ActiveXControl	

Wir werden aber zunächst bei der schönen graphischen Darstellung unserer Funktion

$$z = f(x,y) = \frac{\sin\left(\sqrt{x^2+y^2}\right)+\frac{1}{1000}}{\sqrt{x^2+y^2}+\frac{1}{1000}}$$

bleiben und einen anderen Bereich der GUI-Programmierung kennenlernen: das Einfügen von Menüs.

Menüs

Ein Menü (engl. „menu“) dient zur Auswahl von Befehlen. Die Befehle sind in sogenannten Menüfeldern eingetragen, die durch Anklicken in Form einer sogenannten Menüspalte aufgeklappt werden. Ein solches Menü lernten wir bereits kennen: das Pop-up-Menü, das innerhalb des GUI's placiert wurde.

Wir unterscheiden hier zwei Arten von Menüs:

- Menu Bars („Menüleiste“)
- Contextmenu („Kontext-Menü“).

Menu Bars

Auf Deutsch meint man damit „Menüleiste“. Diese befindet sich im Kopf des GUI's und enthält Überschriften, die durch Anklicken die dazugehörigen Menüfelder aufklappen lassen. Anklicken eines Eintrags in einem Menüfeld gibt dann entweder eine weitere Spalte von Untermenüs aus oder löst eine an den Befehl gebundene Aktion im GUI aus.

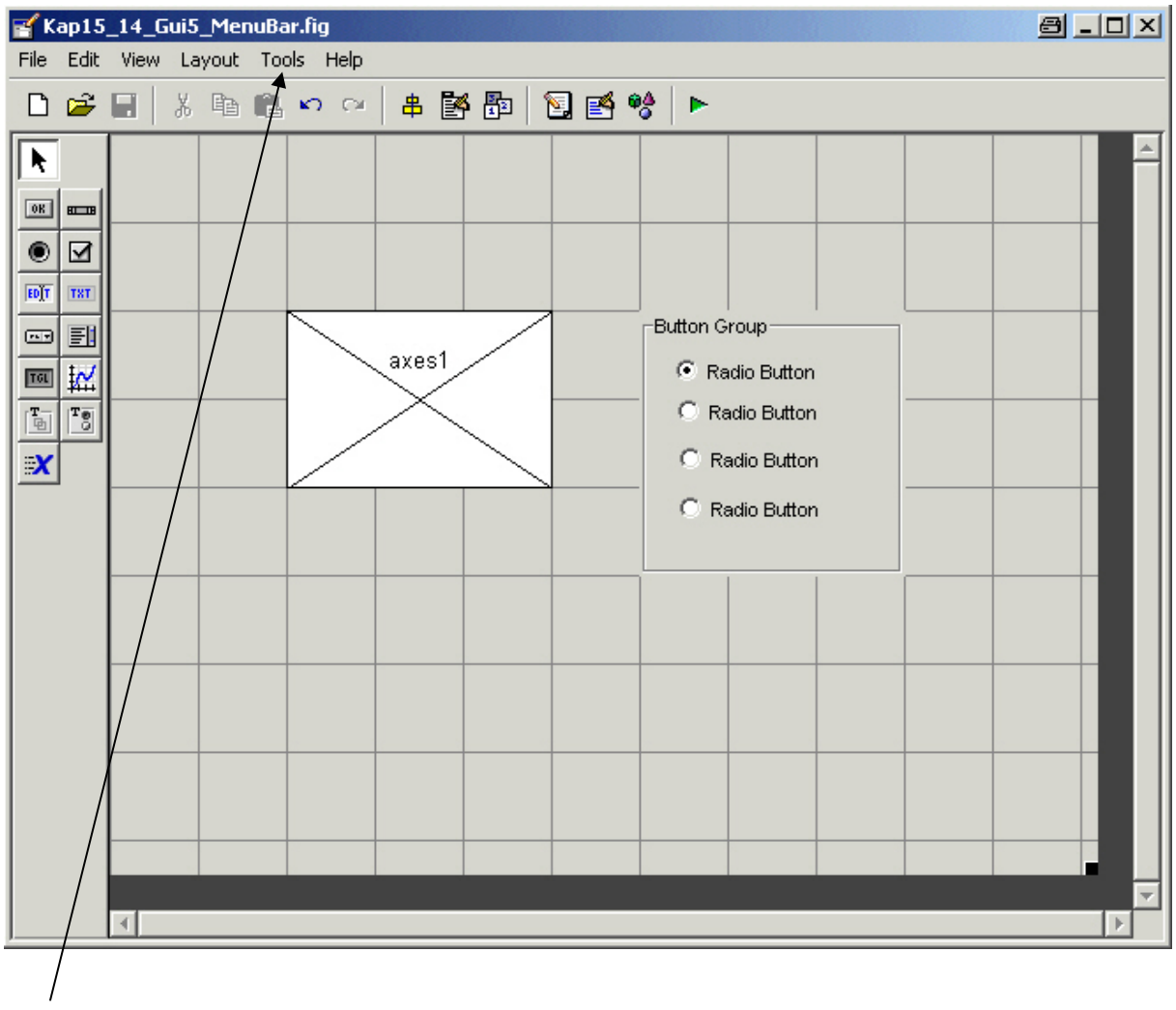
Context menu

Einfach übersetzt „Kontext Menü“. Dies ist nicht sichtbar im GUI. Es wird sichtbar, wenn es durch rechten Mausklick auf ein selektiertes Objekt aktiviert wird. Es öffnet sich dann ein Fenster mit der Spalte der Menüfelder, die ebenfalls durch Anklicken aktiviert werden.

Nachfolgend wollen wir unser bereits vorgestelltes Beispiel der 3D-Darstellung einer Kurve zunächst mit einer Menüleiste und anschließend mit einem Kontext Menü ausstatten.

Zunächst die Menüleiste:

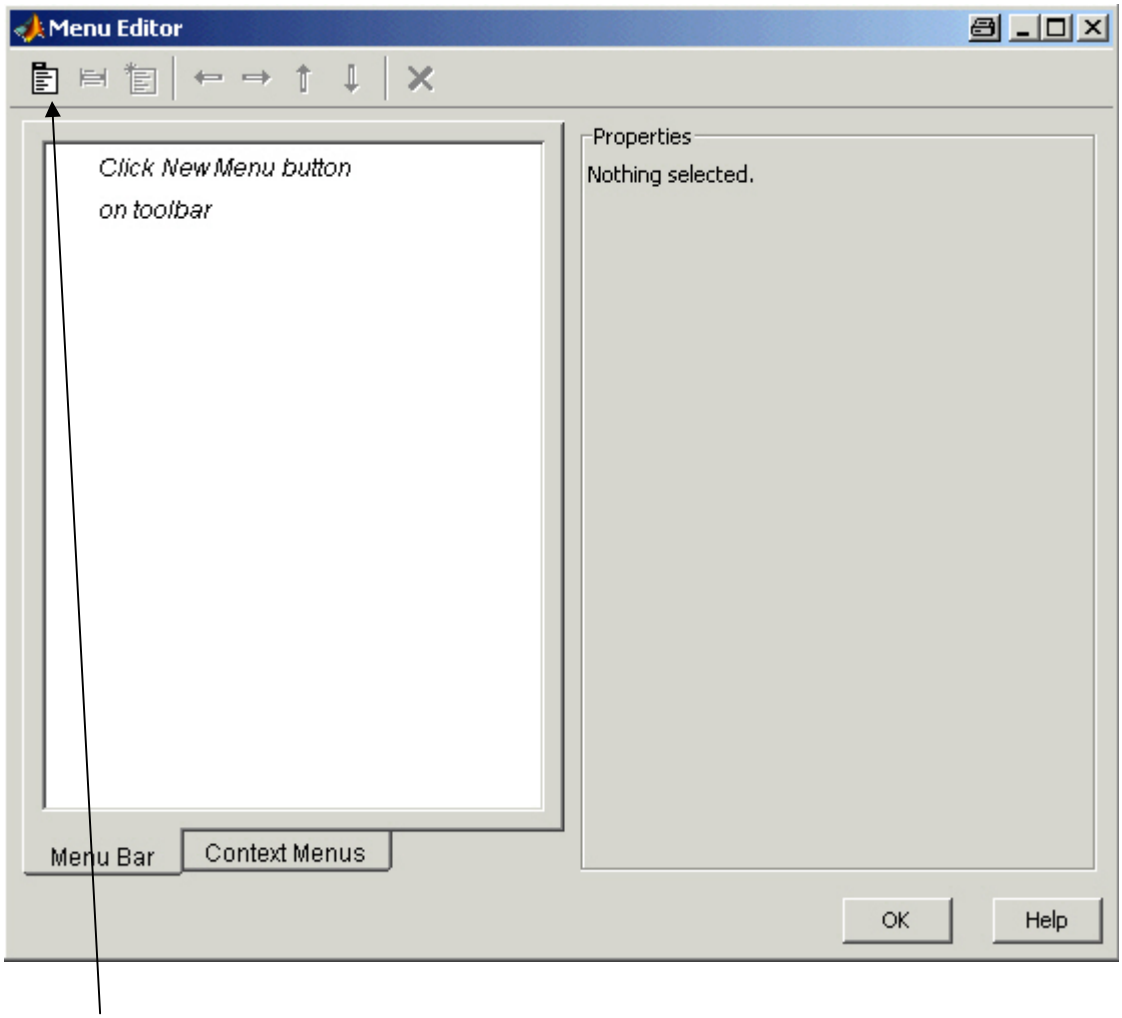
Wir legen einen neuen GUI an, in den wir das Graphik-Objekt und vier in einer Gruppierung zusammengefasste Radio-Buttons eintragen. Dies speichern wir unter „`Kap15_14_Gui5_MenuBar.fig`“ ab.



„Tools“ -> „Menu Editor ...“

In der dazugehörigen Menüleiste (oh ja, da ist bereits eine!) wählt man „Tools“ aus. Es öffnet sich die Menüfelder, unter denen der Eintrag „Menu Editor...“ ausgewählt wird.

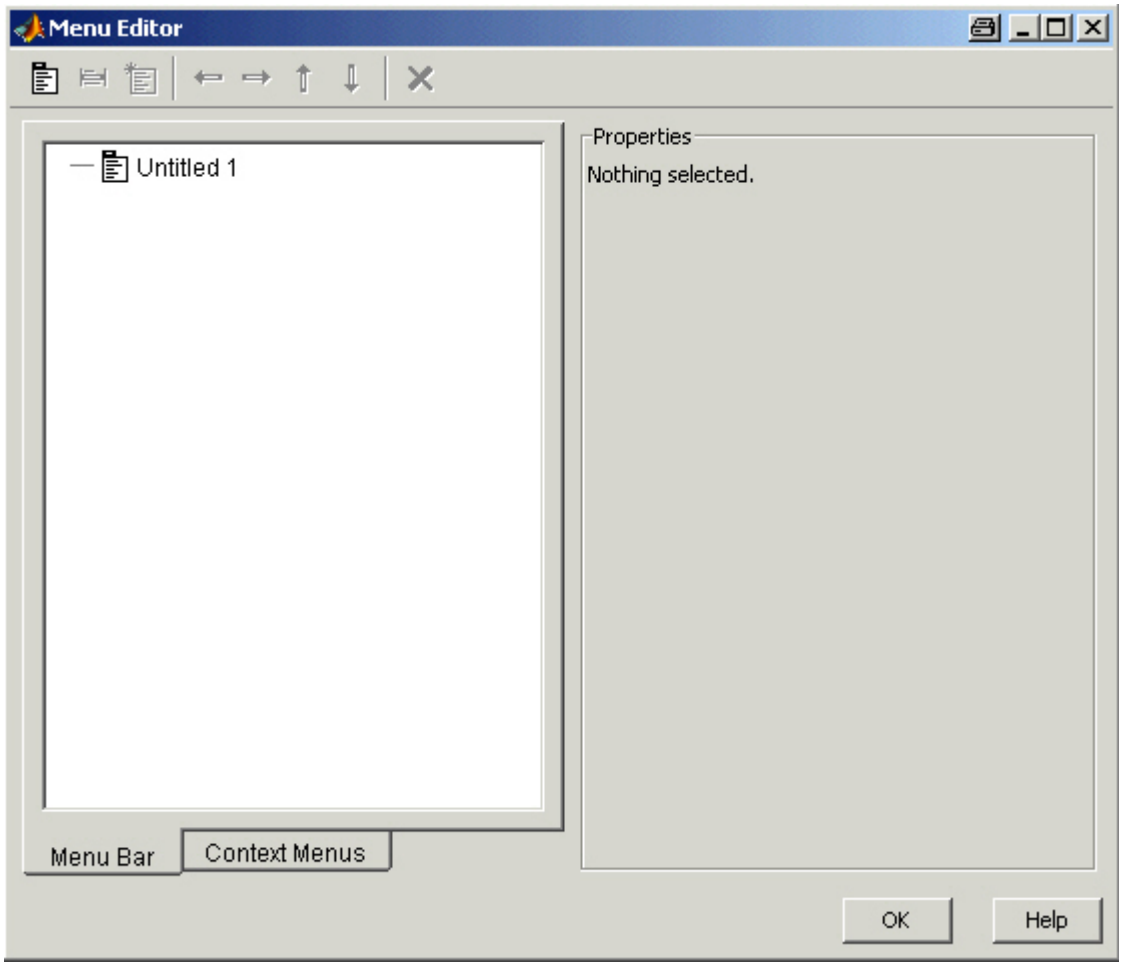
Dies startet den sogenannten Menü-Editor, der sich in Form eines neu geöffneten Fensters präsentiert:



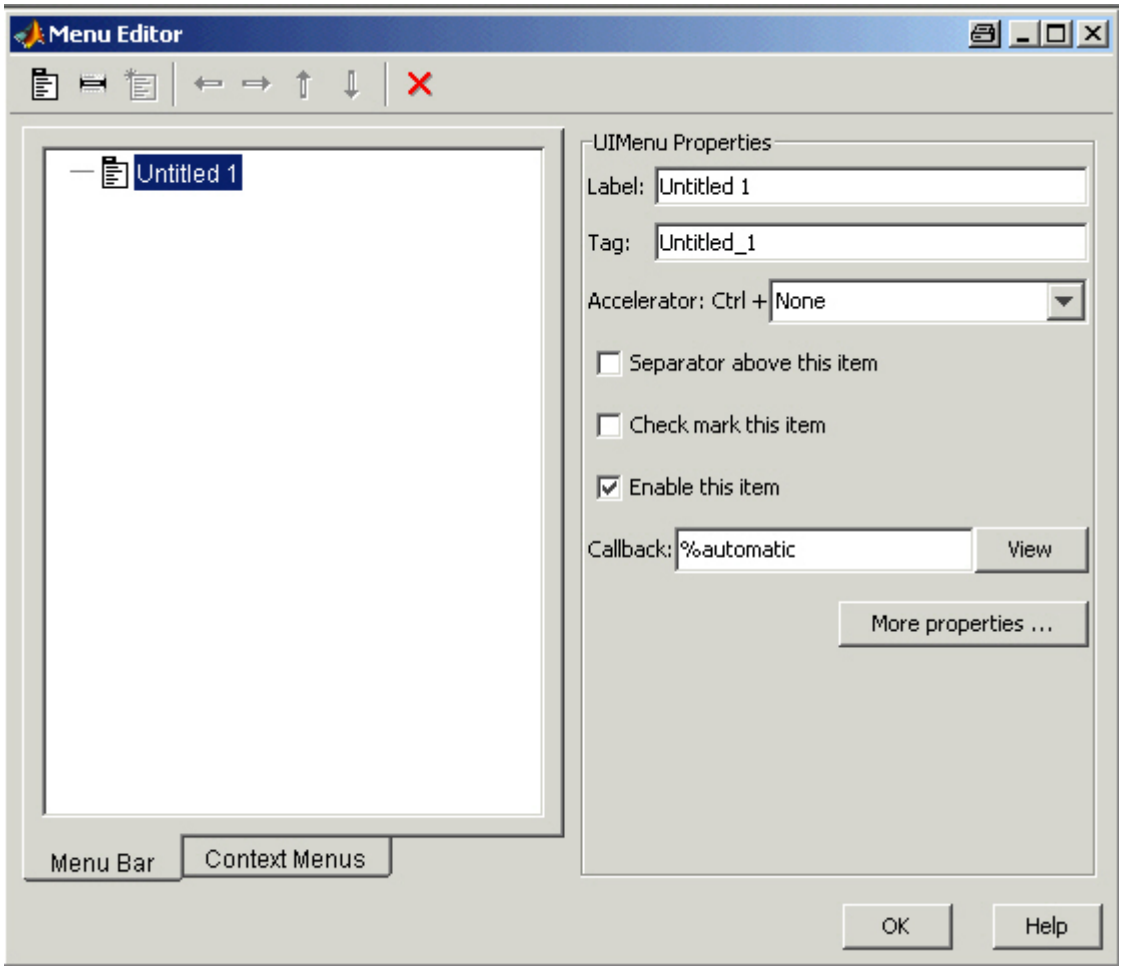
Icon zur Erstellung eines neuen Menüs

Aufgeschlagen ist (am unteren Tab-Reiter zu erkennen): „Menu Bar“
In der Überschriftzeile ist ein einziges Icon aktiviert, nämlich das zur Erstellung eines neuen Menüs (er hat auch den Namen „New Menu“). Es handelt sich dabei um einen Knopf, durch dessen Anklicken eine Aktion – nämlich die Erstellung eines neuen Menüs – ausgelöst wird. Rechts (unter „Properties“) steht „Nothing selected“. Dies alles zeigt den absoluten Neubeginn an. Der Text innerhalb der Liste (dargestellt im linken Teilfensters) fordert auf, den Knopf „New Menu“ zu drücken.

Klickt man also diesen an (ist ohnehin der einzige, der angeklickt werden kann), wird das Fenster wie folgt erweitert:



Die vorher leere Liste wurde ersetzt durch einen neuen Eintrag „Untitled1“. Dies ist (bis jetzt) der Name der Überschrift in der Menüzeile des GUI’s – natürlich völlig nichtssagend und bar jeder Funktionalität. Eine Überschrift und und Einträge sollen hinzukommen.
Dazu wählt man mit der Maus den Eintrag „Untitled1“ aus.



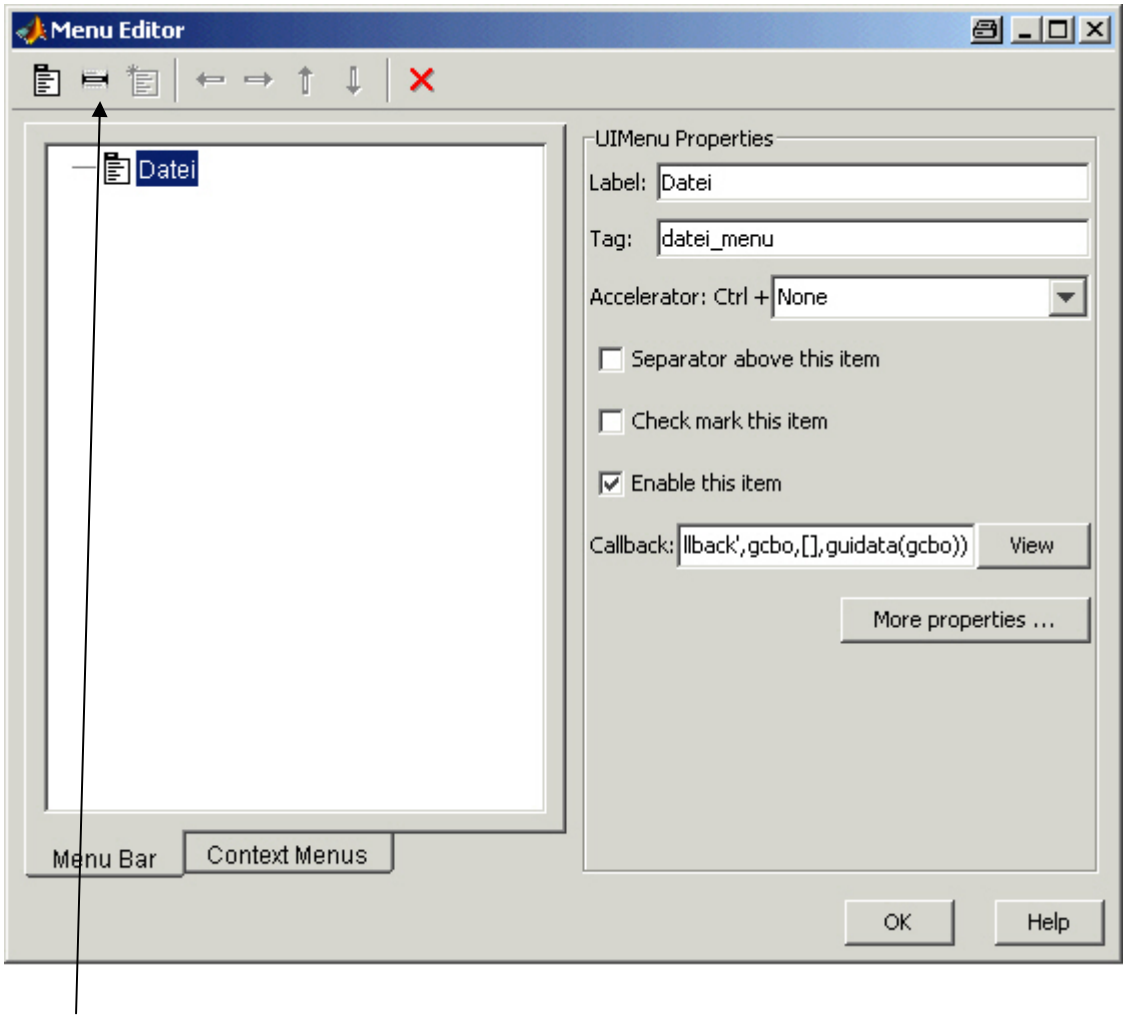
Wie zu ersehen, wird damit auch der rechte Teilbereich zur Aufnahme von Eigenschaften für das Menü geöffnet.

„Label“ bedeutet hier: der nachfolgende Eintrag ist die Überschrift, die in der Menuleiste des GUI erscheint.

„Tag“ ist der Name der Funktion im GUI, unter der dann die dazugehörige Aktion implementiert wird.

Diese Menü-Überschrift soll „Datei“ heißen. Daher tragen wir unter Label „Datei“ ein und unter „Tag“ „datei_menu“.

Es ist übrigens noch in der Menüleiste oben rechts ein rotes Kreuz zu sehen. Dies dient zum Entfernen irrtümlich angelegter Menü-Einträge.

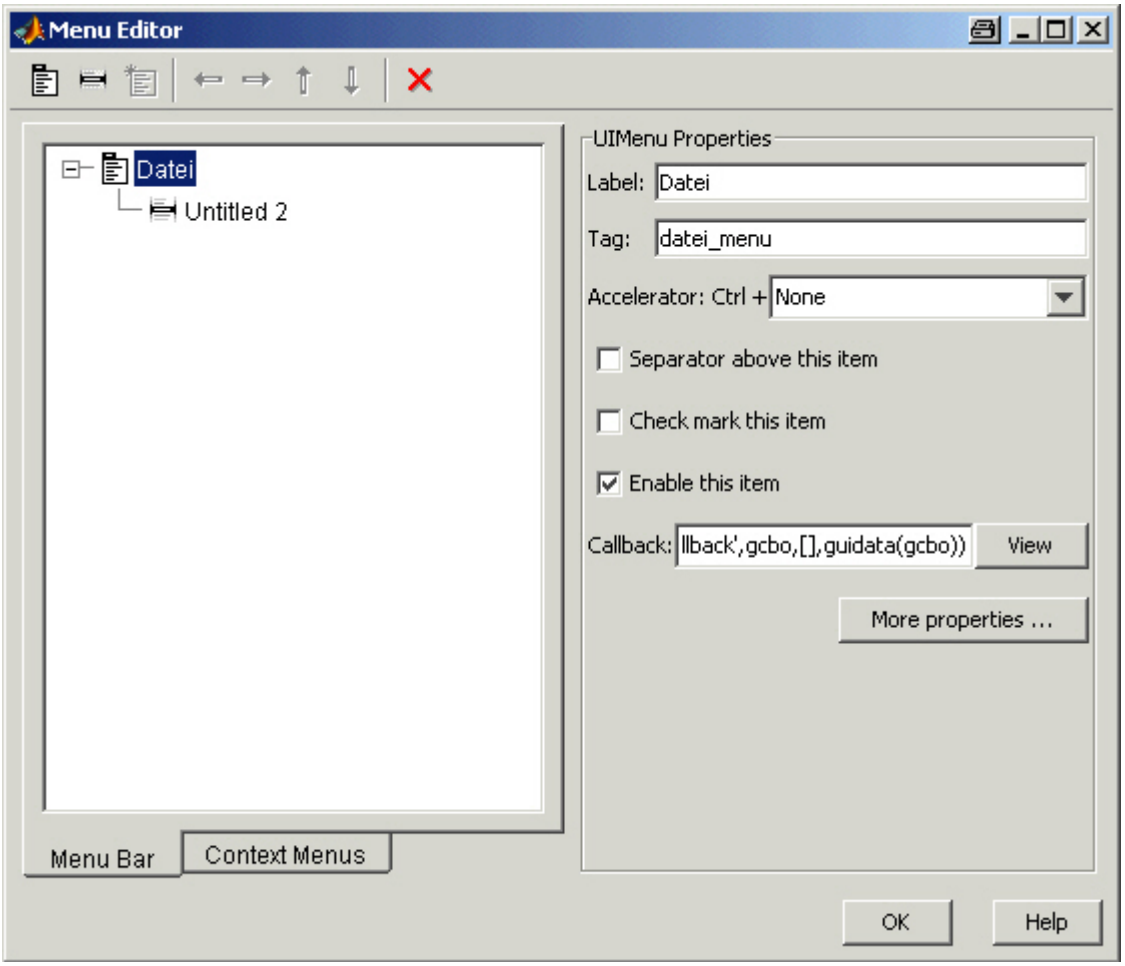


Icon zur Erstellung eines Menufeldes („New Menu Item“)

Wie zu ersehen, wird mit den Einträgen rechts auch die Liste links angepasst:“Untitled1“ ist in „Datei“ umgeändert worden.

Es ist damit nicht nur eine neue Überschrift im GUI erzeugt worden, sondern auch eine komplette Spalte zur Aufnahme der Menüfelder, da diese die eigentlichen Befehle realisieren. Dies geschieht wie folgt:

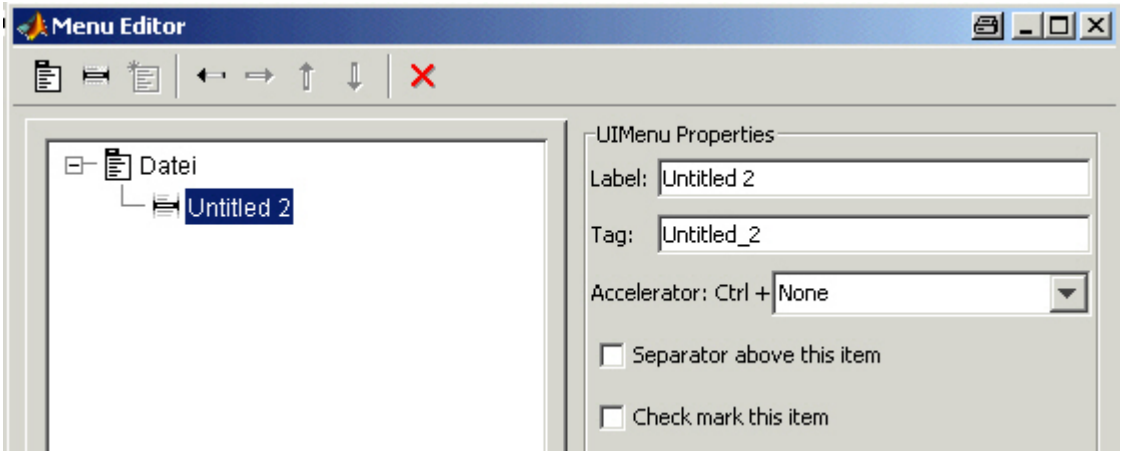
Es ist ein weiterer Icon zur Erstellung eines Menüfeldes aktiviert worden. Er befindet sich rechts neben dem für neue Menü-Überschriften. Diesen klickt man an.



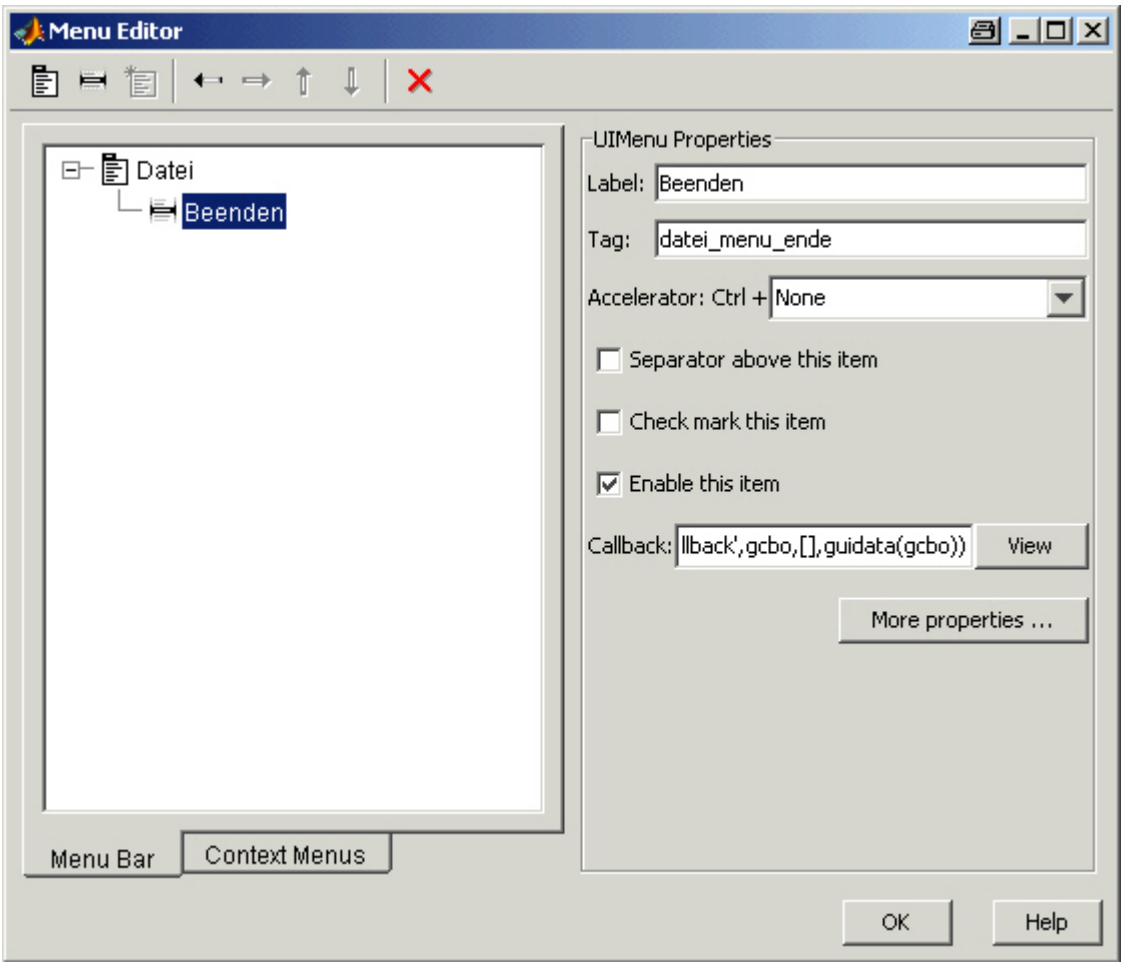
Als Untereintrag von „Datei“ finden wir jetzt „Untitled2“. Dies ist der Eintrag des ersten Menüfeldes. Im GUI würde an dieser Stelle in der Menüleiste links die Überschrift „Datei“ erscheinen, die bei Anwahl dann die Spalte mit dem Eintrag „Untitled2“ öffnet.

Dies ist natürlich auch nicht sonderlich aussagekräftig – abgesehen davon, dass gar keine Aktion folgt.

Die geplante Aktion ist, bei Anklicken des Eintrags den GUI zu schließen. Daher klickt man also „Untitled2“ an:



und trägt in Label und Tag ein:

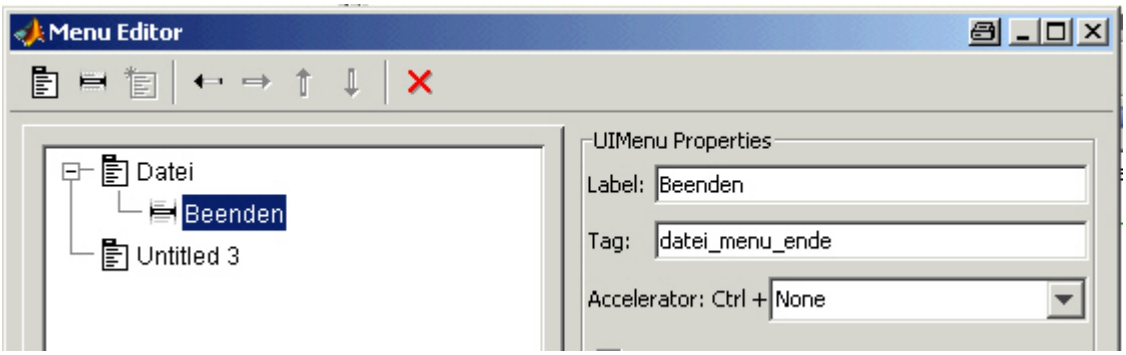


Label: „Beenden“

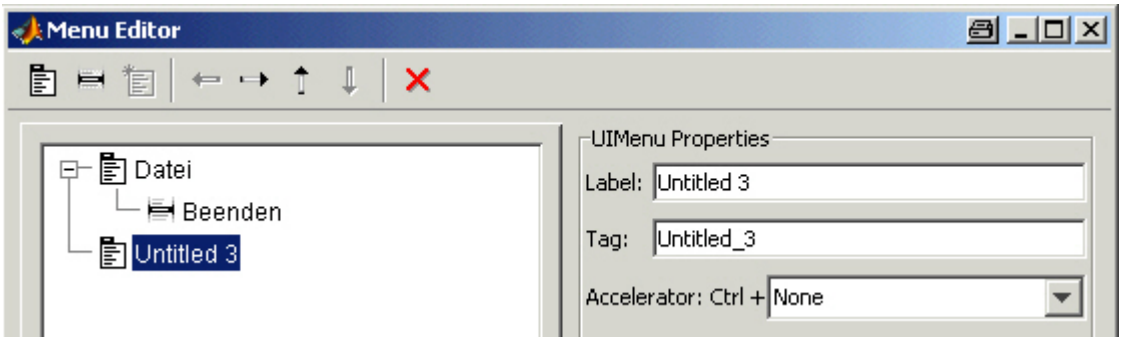
Tag: „datei_menu_ende“

Eine weitere Überschrift soll in der Menüleiste erscheinen: eben diejenige, die die Auswahl der Darstellungsansicht der dreidimensionalen Kurve.

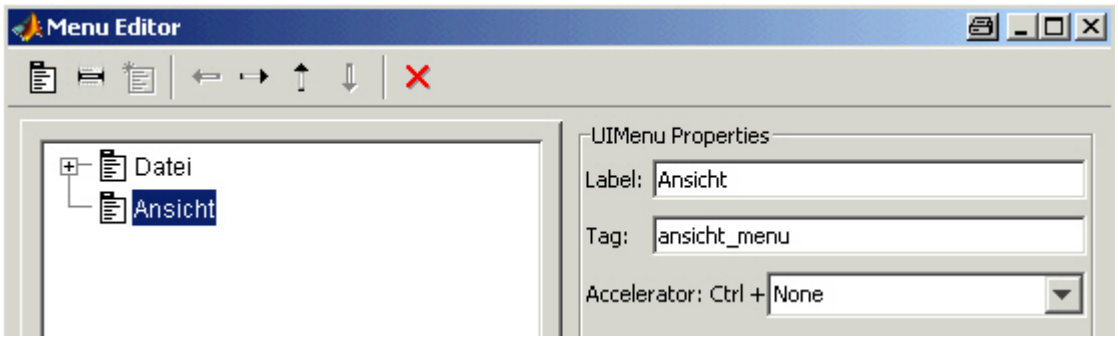
Dazu klickt man wieder den Icon „New Menu“ an (Achtung: es ist wichtig, diesen anzuklicken – sonst erhält man möglicherweise wieder Untereinträge zu „Beenden“ oder wo auch immer).



und klickt jetzt „Untitled3“ an:



Jetzt kann auch dieser Eintrag editiert werden:

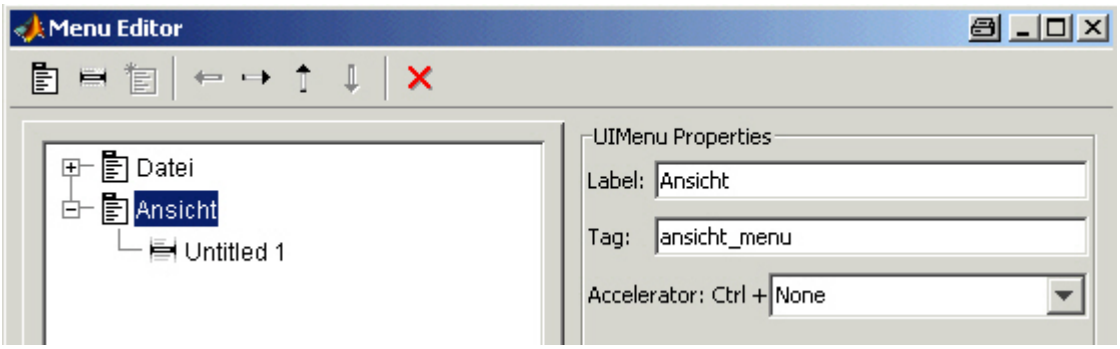


Man trägt ein:

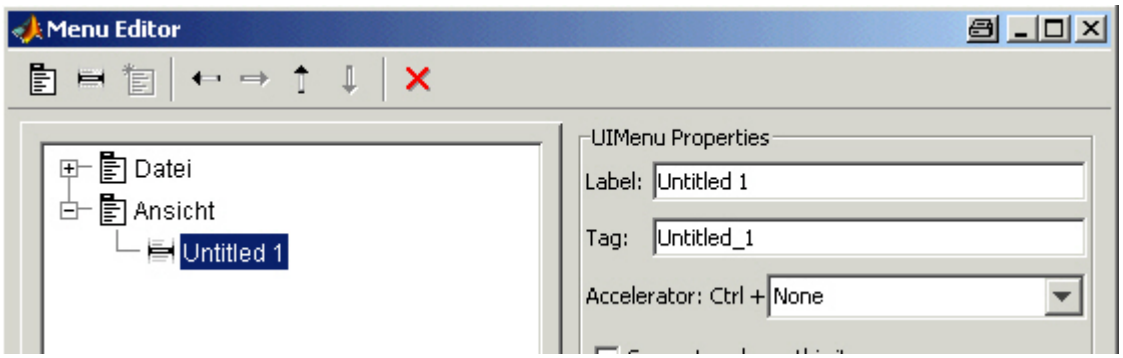
Label: „Ansicht“

Tag: „ansicht_menu“

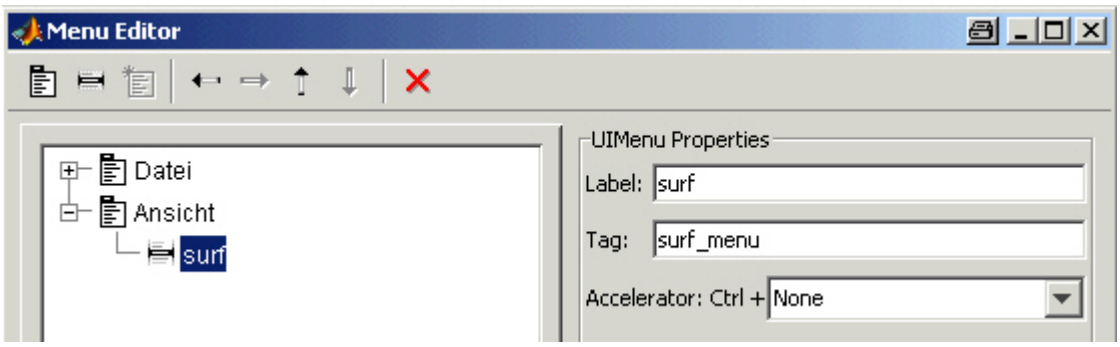
Um für diese Überschrift die Felder zu definieren, klickt man jetzt erneut den Icon „New Menu Item“ an:



Man klickt „Untitled1“ an



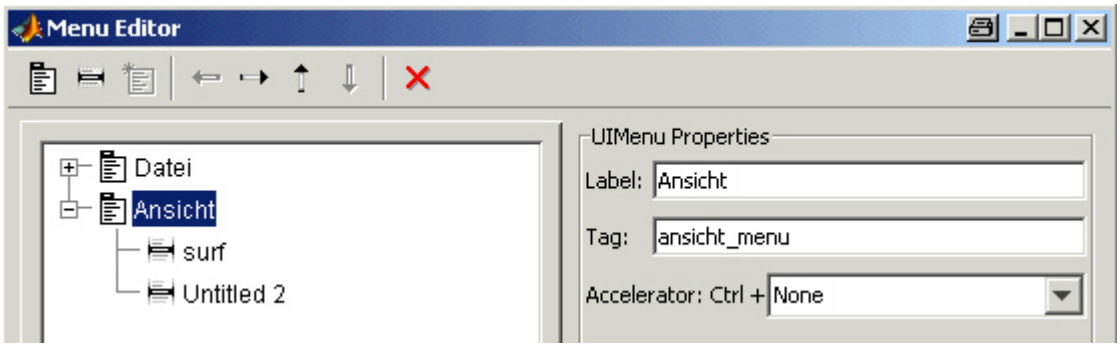
und trägt ein:



Label: „surf“

Tag: „surf_menu“

„Ansicht“ soll noch ein weiteres Menüfeld erhalten (insgesamt übrigens vier). Damit der folgende Eintrag ein Feld von „Ansicht“ ist (und nicht von „surf“), muß jetzt wieder „Ansicht“ angeklickt werden; *dann erst* wird der Icon „New Menu Item“ angeklickt:

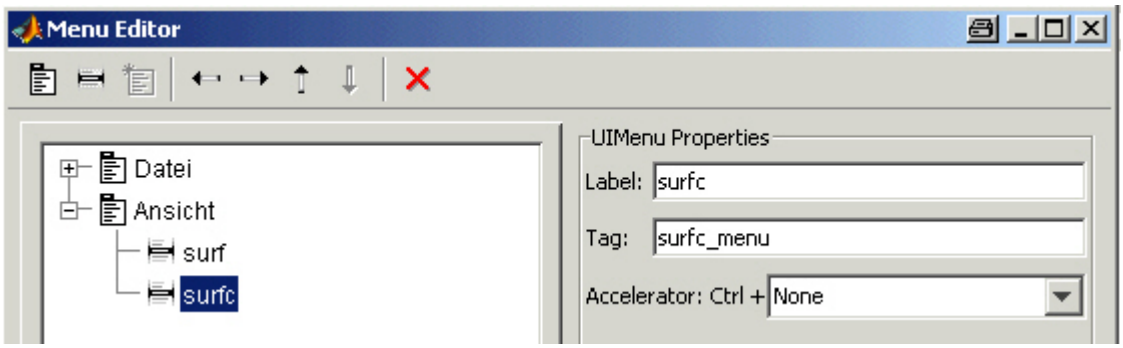


Anklicken von „Untitled2“ und Eintragen

Label: „surfc“

Tag: „surfc_menu“

führt zu



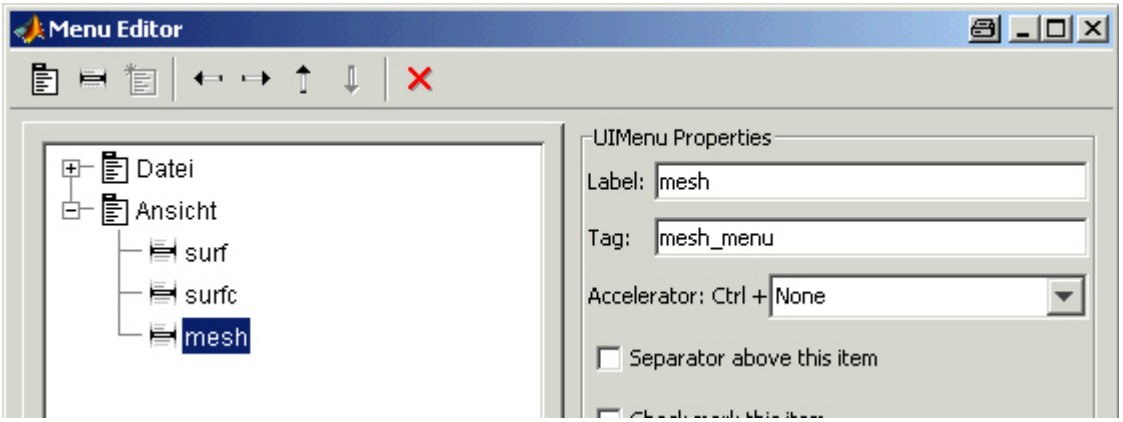
Zum Eintragen des dritten Menüfeldes „mesh“ unter „Ansicht“ geht man analog vor:

Anklicken „Ansicht“

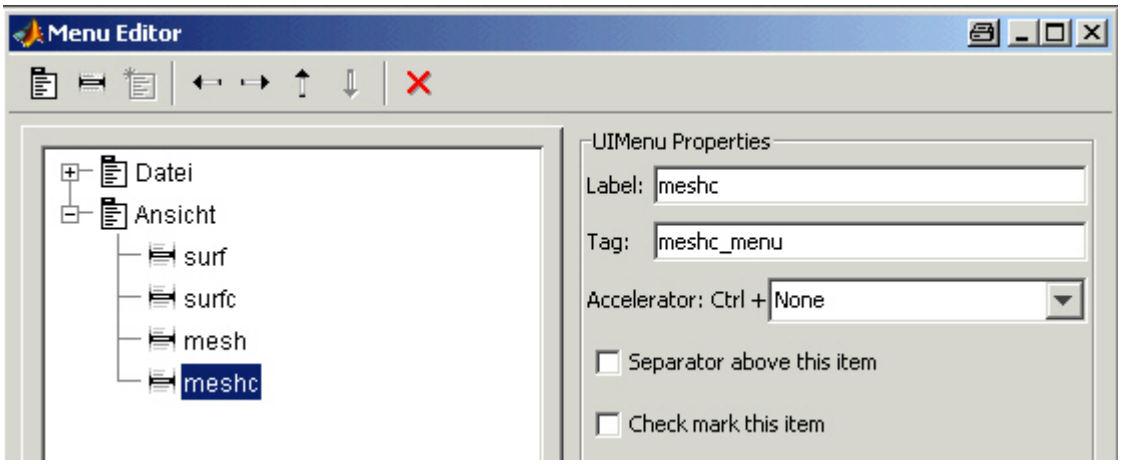
Icon „New Menu Item“

Anklicken „Untitled3“

Eintragen Label „mesh“, Tag „mesh_menu“:



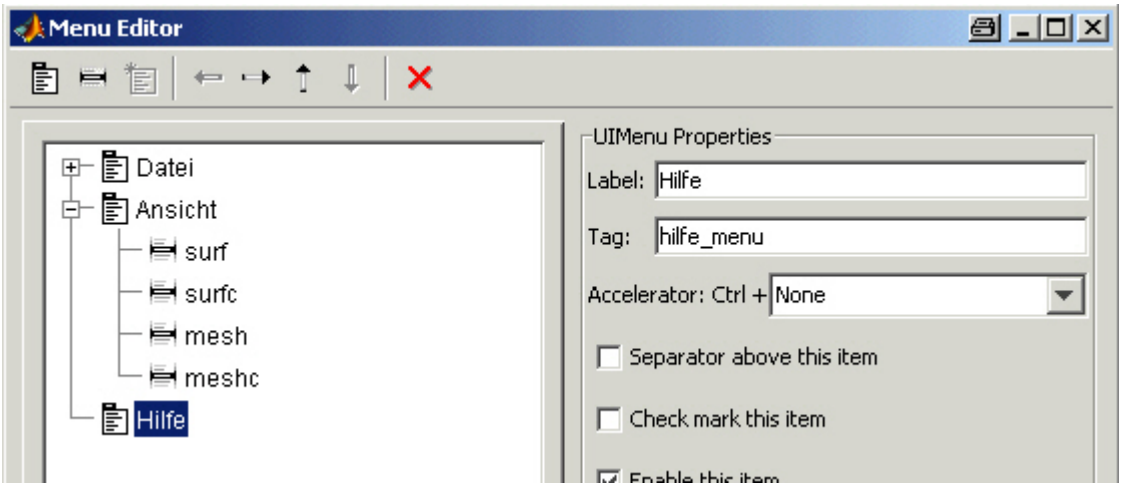
Entsprechend wird auch das vierte Menüfeld „meshc“ eingefügt:



Label: „meshc“

Tag: „meshc_menu“

Um eine neue Überschrift „Info“ einzufügen, klickt man jetzt erneut den Icon „New Menu“ an, klickt dann auf „Untitled5“ und setzt wieder:



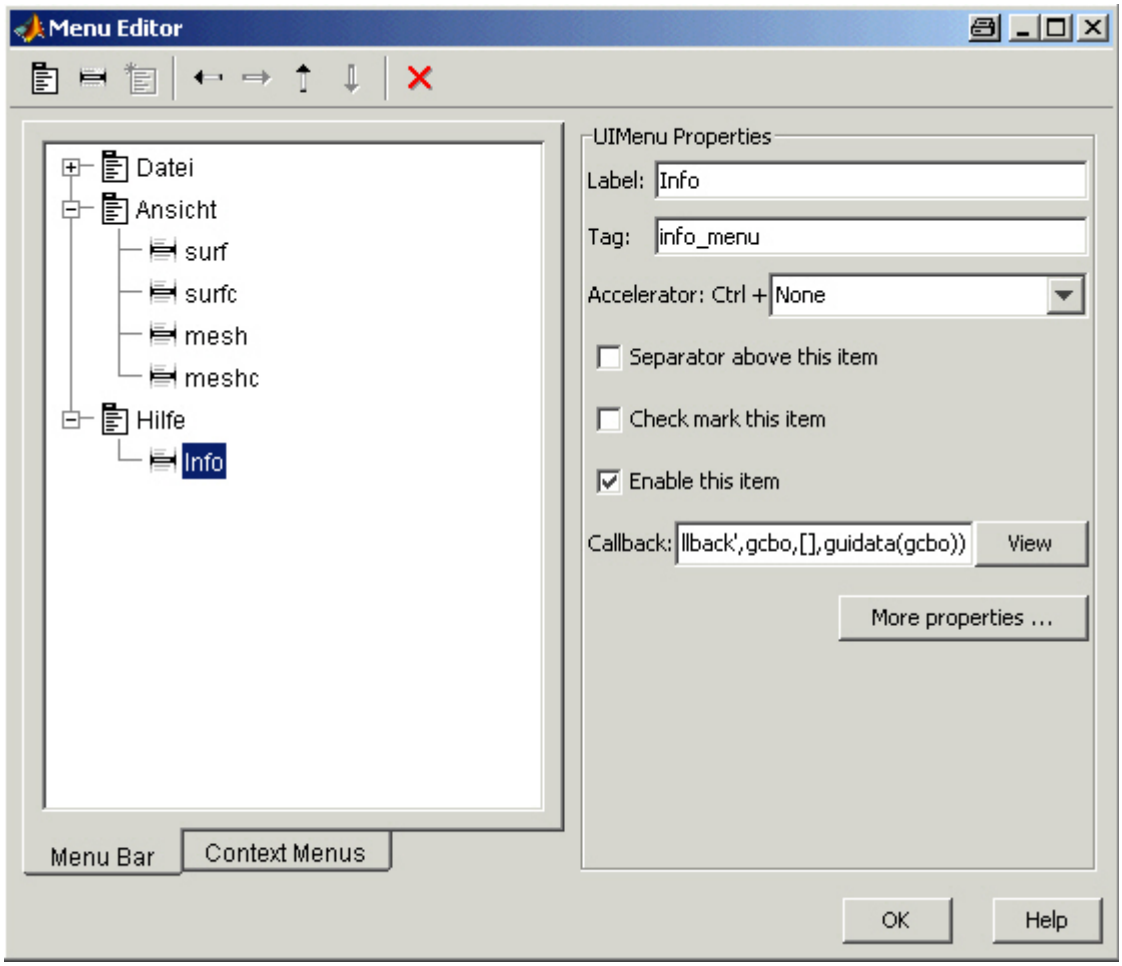
Label: „Hilfe“

Tag: „hilfe_menu“

Für dieses ist dann ebenfalls ein Feld vorgesehen, nämlich die Ausgabe einer Information über das Programm. Der Befehl, der die Ausgabe auslöst, soll „Info“ heißen. Nach bekannter Weise wird dieser wie folgt erzeugt

Anklicken Icon „New Menu Item“

Anklicken von „Untitled6“



Label: „Info“

Tag: „info_menu“

Dies ergibt dann (endlich) obiges Bild des Editors.

Damit sind die Vorarbeiten abgeschlossen – mit „OK“ kann der Editor geschlossen werden. „Vorarbeiten“ heißt, dass ja noch gar keine Aktion erfolgt, wenn diese Einträge angewählt werden. Dies muß jetzt erst noch im erzeugten Programm „`Kap15_14_Gui5_MenuBar.m`“ implementiert werden.

Zunächst übernehmen wir die bereits erstellten Funktionsteile des vorangegangenen Beispiels, insbesondere erstellen wir auch hier (durch Anklicken der Gruppierung „Button Group“ und Auswahl von „View Callbacks“ -> „SelectionChangeFcn“ die Umschaltfunktion für die Radio-Buttons, hier also `uipanel1_SelectionChangeFcn(hObject, eventdata, handles)`.

Der Code in „Kap15_14_Gui5_MenuBar.m“ sieht also genauso aus wie in „Kap15_13_Gui5_GraphRadioButton.m“ und ist daher hier nicht aufgeführt. Allerdings enthält die Datei noch zusätzlich (dank der Aktionen im Menü-Editor) weitere Funktionen:

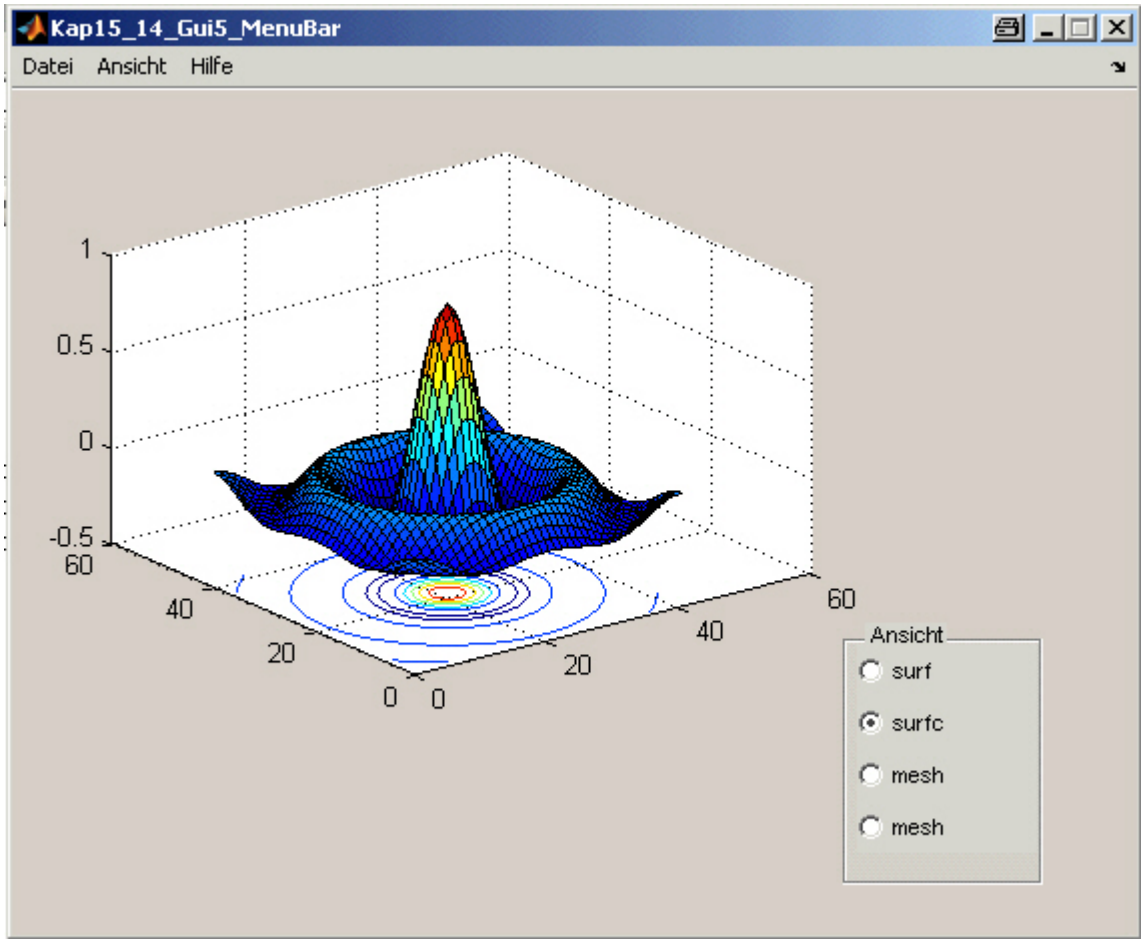
```
function datei_menu_Callback(hObject, eventdata, handles)
function datei_menu_ende_Callback(hObject, eventdata, handles)
function ansicht_menu_Callback(hObject, eventdata, handles)
function surf_menu_Callback(hObject, eventdata, handles)
function surfc_menu_Callback(hObject, eventdata, handles)
function mesh_menu_Callback(hObject, eventdata, handles)
function meshc_menu_Callback(hObject, eventdata, handles)
function hilfe_menu_Callback(hObject, eventdata, handles)
function info_menu_Callback(hObject, eventdata, handles)
```

Diese werden aufgerufen, wenn ein Menüfeld angeklickt wird.

Noch hat ein Aufruf

```
>> Kap15_14_Gui5_MenuBar
>>
```

nur die Folge, daß zwar in der Menüzeile die Überschriften “Datei”, „Ansicht“, „Hilfe“ erscheinen und bei Anklicken entsprechen die Spalte mit den Feldern ausgeben, aber sonst noch keine Aktion erfolgt:



Dies soll nun nachgeholt werden.

Für die mit den Überschriften „Datei“, „Ansicht“, „Hilfe“ sind keine Aktionen vorgesehen; daher bleiben die dazugehörigen Rückruffunktionen

```
function datei_menu_Callback(hObject, eventdata, handles)
function ansicht_menu_Callback(hObject, eventdata, handles)
function hilfe_menu_Callback(hObject, eventdata, handles)
```

leer.

Hingegen soll bei Aufruf „Datei“ -> „Beenden“ der GUI geschlossen werden. Daher wird in `datei_menu_ende_Callback` eingetragen:

```
delete(handles.figure1);
```

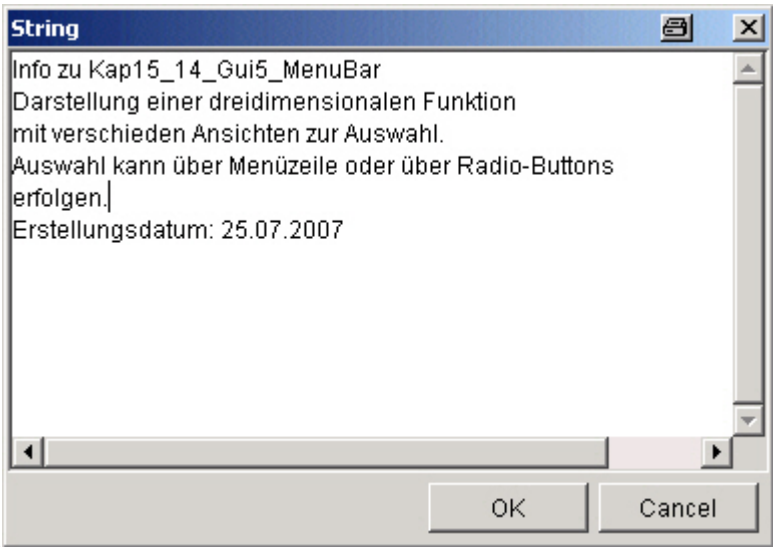
In die vier Rückruffunktionen

```
function surf_menu_Callback(hObject, eventdata, handles)
function surfc_menu_Callback(hObject, eventdata, handles)
function mesh_menu_Callback(hObject, eventdata, handles)
function meshc_menu_Callback(hObject, eventdata, handles)
```

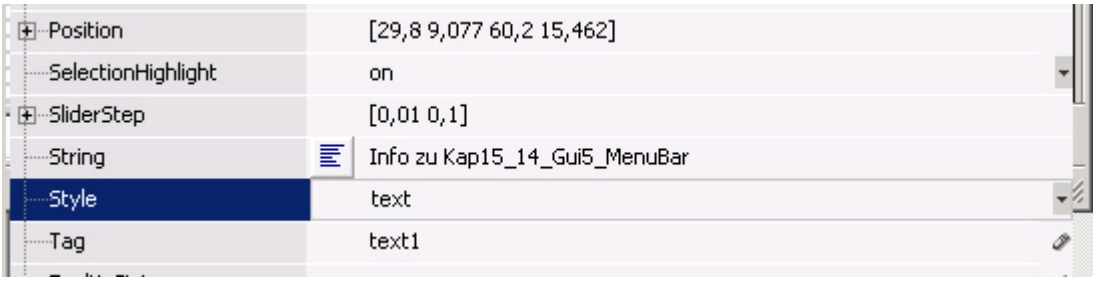
wird je nach Ansichtstyp die Zeichenfunktion `surf`, `surfc`, `mesh` oder `meshc` aufgerufen. Außerdem wird dies mit den Radio-Buttons synchronisiert, indem dessen Wert auf 1 gesetzt wird.

Für die Info-Rückruf-Funktion wird mittels GUIDE ein neuer GUI erzeugt:

In den neuen GUI setzen wir ein großzügig bemessenes Textfeld („static text“). Mit rechtem Mausklick öffnen wir das Kontextmenü und gehen in den Property-Inspector. Wir klicken den Eintrag „String“an. Es öffnet sich ein weiteres und trägt dort einen (wie auch immer) geeigneten Text ein:



Zusätzlich setzen wir noch die Eigenschaft „Style“ auf „text“ (wenn nicht schon automatisch geschehen):



Den GUI speichern wir unter „`Gui5_Info.fig`“.

Und in `info_menu_Callback` geben wir einfach den Befehl „`Gui5_Info`“ ein.

Damit lautet der vollständige Programmtext:

```
function varargout = Kap15_14_Gui5_MenuBar(varargin)
% KAP15_14_GUI5_MENUBAR M-file for Kap15_14_Gui5_MenuBar.fig
%      KAP15_14_GUI5_MENUBAR, by itself, creates a new
%      KAP15_14_GUI5_MENUBAR or raises the existing singleton*.
%
%      H = KAP15_14_GUI5_MENUBAR returns the handle to a new
%      KAP15_14_GUI5_MENUBAR or the handle to the existing singleton*.
%
%      KAP15_14_GUI5_MENUBAR('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in KAP15_14_GUI5_MENUBAR.M
%      with the given input arguments.
%
%      KAP15_14_GUI5_MENUBAR('Property','Value',...) creates a new
%      KAP15_14_GUI5_MENUBAR or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_14_Gui5_MenuBar_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to
%      Kap15_14_Gui5_MenuBar_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_14_Gui5_MenuBar

% Last Modified by GUIDE v2.5 25-Jul-2007 10:32:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_14_Gui5_MenuBar_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_14_Gui5_MenuBar_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_14_Gui5_MenuBar is made visible.
function Kap15_14_Gui5_MenuBar_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_14_Gui5_MenuBar (see VARARGIN)

% Choose default command line output for Kap15_14_Gui5_MenuBar
handles.output = hObject;

x = -10 : 0.5 : 10;
y = x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.*xx + yy.*yy);
handles.z = (sin(r) + 0.001)./(r + 0.001);
surf(handles.z);

set(handles.axes1, 'Position', [10 10 70 20]);

set(handles.uipanel1, 'Title', 'Ansicht');
set(handles.uipanel1, 'Position', [83 2 20 10]);

set(handles.radiobutton1, 'String', 'surf');
set(handles.radiobutton1, 'Position', [1 7  18 2]);
set(handles.radiobutton2, 'String', 'surfc');
set(handles.radiobutton2, 'Position', [1 5 18 2]);
set(handles.radiobutton3, 'String', 'mesh');
set(handles.radiobutton3, 'Position', [1 3 18 2]);
set(handles.radiobutton4, 'String', 'mesh');
set(handles.radiobutton4, 'Position', [1 1 18 2]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_14_Gui5_MenuBar wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_14_Gui5_MenuBar_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function datei_menu_Callback(hObject, eventdata, handles)
% hObject    handle to datei_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
```

```
function datei_menu_ende_Callback(hObject, eventdata, handles)
% hObject    handle to datei_menu_ende (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
delete(handles.figure1);

% --------------------------------------------------------------------
function ansicht_menu_Callback(hObject, eventdata, handles)
% hObject    handle to ansicht_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function surf_menu_Callback(hObject, eventdata, handles)
% hObject    handle to surf_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
surf(handles.z);
set(handles.radiobutton1, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function surfc_menu_Callback(hObject, eventdata, handles)
% hObject    handle to surfc_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
surfc(handles.z);
set(handles.radiobutton2, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function mesh_menu_Callback(hObject, eventdata, handles)
% hObject    handle to mesh_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mesh(handles.z);
set(handles.radiobutton3, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function meshc_menu_Callback(hObject, eventdata, handles)
% hObject    handle to meshc_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
meshc(handles.z);
set(handles.radiobutton4, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function hilfe_menu_Callback(hObject, eventdata, handles)
% hObject    handle to hilfe_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function info_menu_Callback(hObject, eventdata, handles)
% hObject    handle to info_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Gui5_Info;
```

```
% --------------------------------------------------------------------
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(hObject,'Tag')   % Get Tag of selected object
    case 'radiobutton1'
        % Code for when radiobutton1 is selected.
        surf(handles.z);
    case 'radiobutton2'
        % Code for when radiobutton2 is selected.
        surfc(handles.z);
    case 'radiobutton3'
        % Code for when togglebutton1 is selected.
        mesh(handles.z);
    case 'radiobutton4'
        % Code for when togglebutton2 is selected.
        meshc(handles.z);
    otherwise
        % Code for when there is no match.
        warning('Fall bei Aufruf Radio Buttons vergessen!');
        fprintf('tag = %s\n', get(h.Object,'Tag'));
end
```

Nach dem Aufruf

```
>> Kap15_14_Gui5_MenuBar
>>
```

kann jetzt zusätzlich über die entsprechenden Menüfelder die Ansicht umgeschaltet werden. Die Radio-Buttons werden ebenfalls mit aktualisiert.

Öffnet man „Datei“ -> „Beenden“, wird der GUI geschlossen.

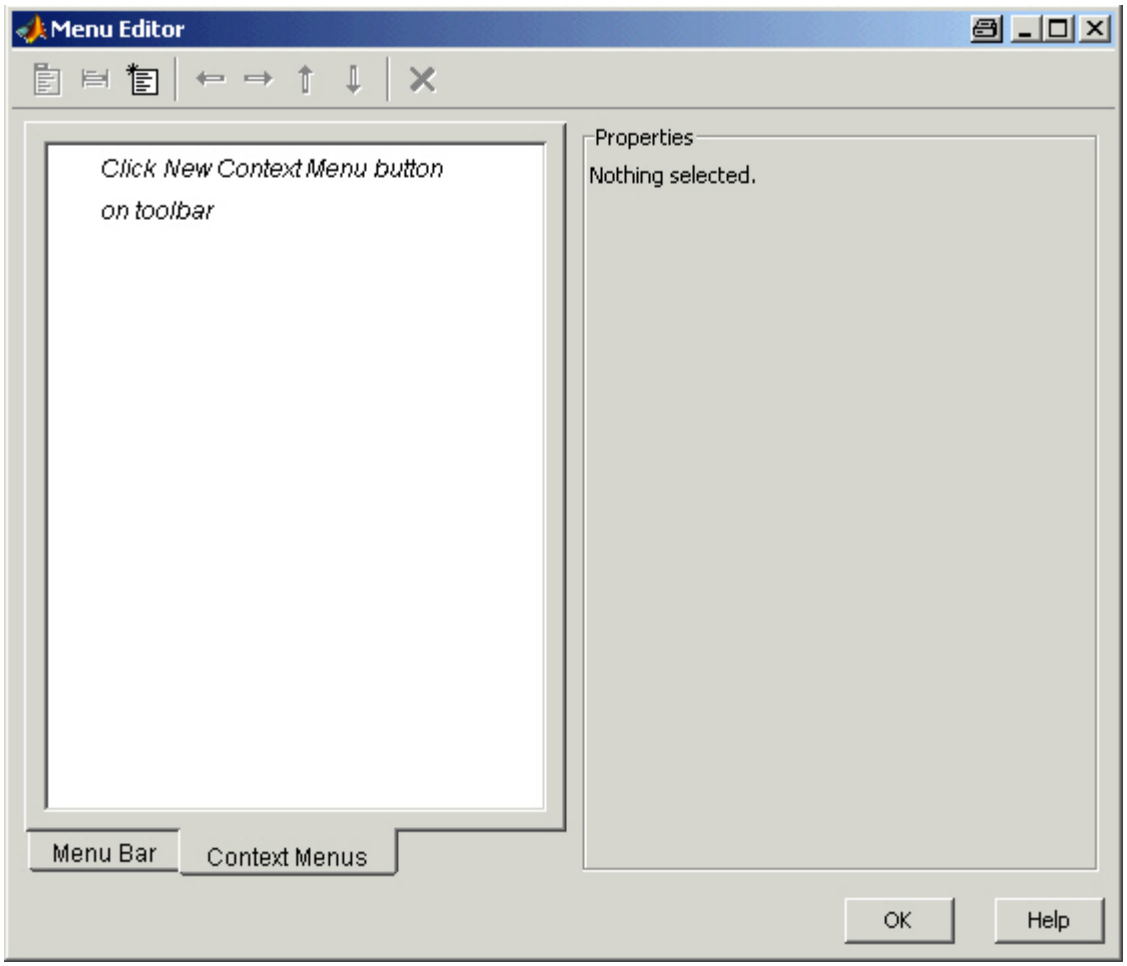
Öffnet man „Hilfe“ -> „Info“, erscheint in einem separaten GUI der mäßig schöne Info-Text. Da dieser GUI unabhängig vom ersten ist, bleibt der Info-GUI auch noch dann offen, wenn der Ansichts-GUI bereits geschlossen worden ist.

Jetzt mit Kontext-Menü:

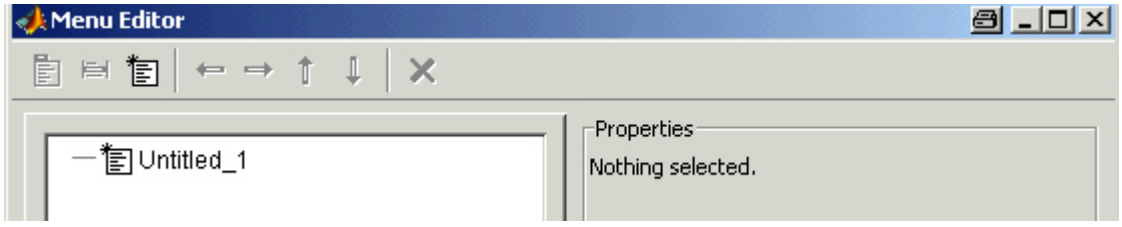
Wenn das Graphik-Objekt mit der rechten Maustaste angeklickt wird, soll ein Kontextmenü mit den Feldern „surf“, „surfc“, „mesh“ und „meshc“ erscheinen. Die Abfolge der Erzeugung verläuft mittels des Menü-Editors inGUIDE ähnlich wie bei einer Menüleiste.

Zunächst erstellen wir von „Kap15_14_Gui5_MenuBar“eine Kopie, die wir „Kap15_15_Gui5_ContextMenu“ nennen (am einfachsten in GUIDE den Tab „Open existing GUI“ anwählen, dann den Eintrag „Kap15_14_Gui5_MenuBar“ auswählen und diesen unter „Kap15_15_Gui5_ContextMenu“ abspeichern).

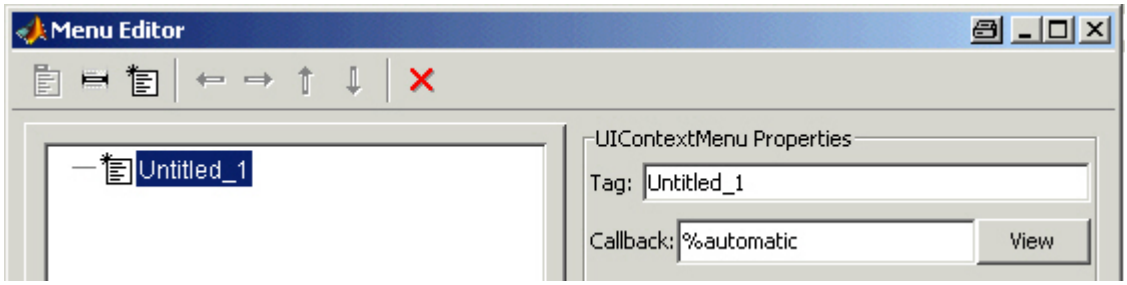
In GUIDE rufen wir wieder den Menü-Editor auf („Tools“ -> „Menu Editor“), aber diesmal mit dem Tab „Context Menu“).



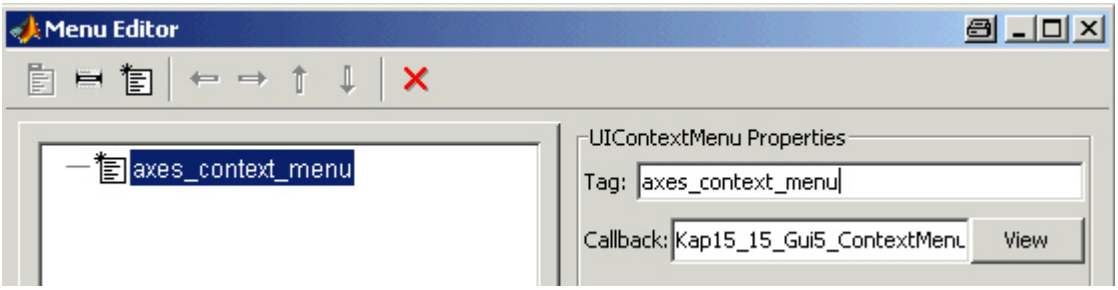
Diesmal ist der dritte Icon von rechts aktiv – er steht für „New Context Menu“. Diesem klicken wir an:



es erscheint jetzt in der linken Auswahlliste der Eintrag „Untitled1“ (also im Prinzip wie bei einem Menü aus einer Menüleiste). Anklicken von „Untitled_1“ liefert:



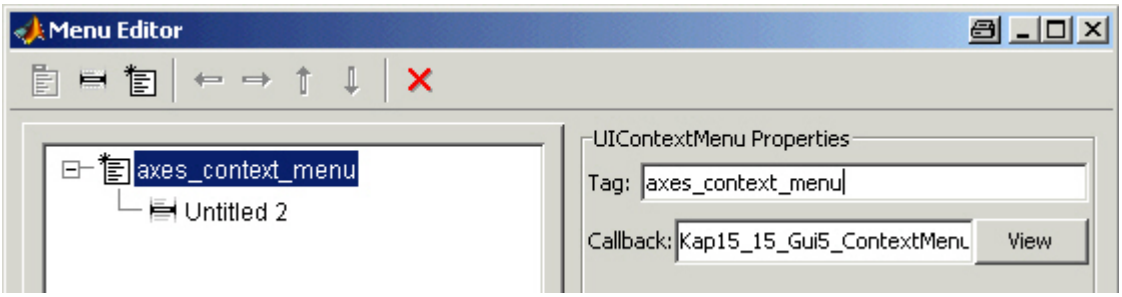
im rechten Teil öffnen sich editierbare Textfelder. Das Feld „Tag“ wird mit einem Namen belegt:



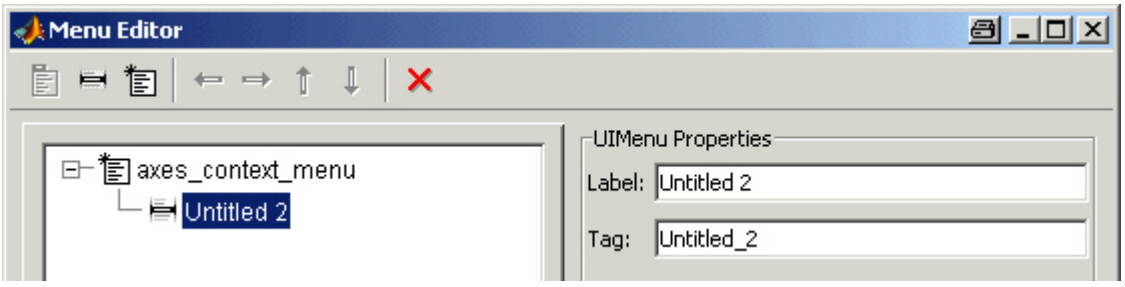
Tag: „axes_context_menu“
Wie zu ersehen, werden damit auch die Einträge in „Callback“ und in der Auswahlliste aktualisiert.

„axes_context_menu“ ist der Name, unter dem dieses Menü identifiziert wird – wir werden ihn noch benötigen, um ihn mit dem Graphik-Objekt zu verbinden. Ein Feld „Label“ gibt es hier nicht – es wird ja keine Überschrift benötigt, die bei Anklicken des Objekts erscheint.
Dem „axes_context_menu“ werden Menufelder zugeordnet.

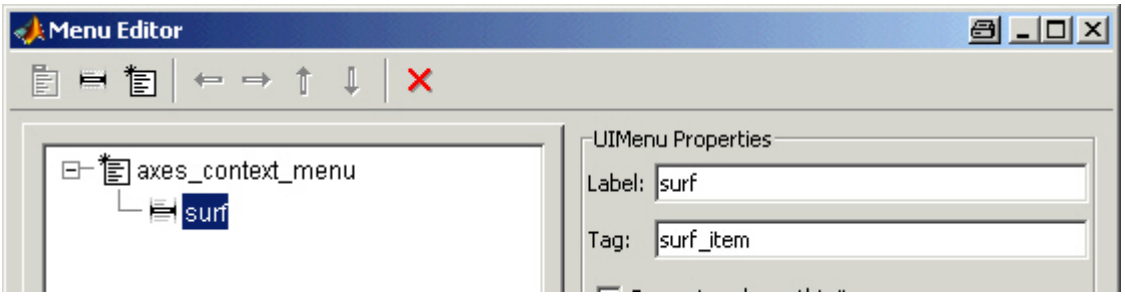
Dazu klickt man wieder den zweiten (jetzt aktiven) Icon von rechts an (das ist wie bei den Menüzeilen der Icon für „New Menu Item“).



Anklicken von „Untitled2“ öffnet wieder zwei editierbare Texteingabefelder:



Label ist wieder der Eintrag im Menüfeld und Tag der Name, mit dem das Programm den Eintrag identifiziertr. Wir setzen:



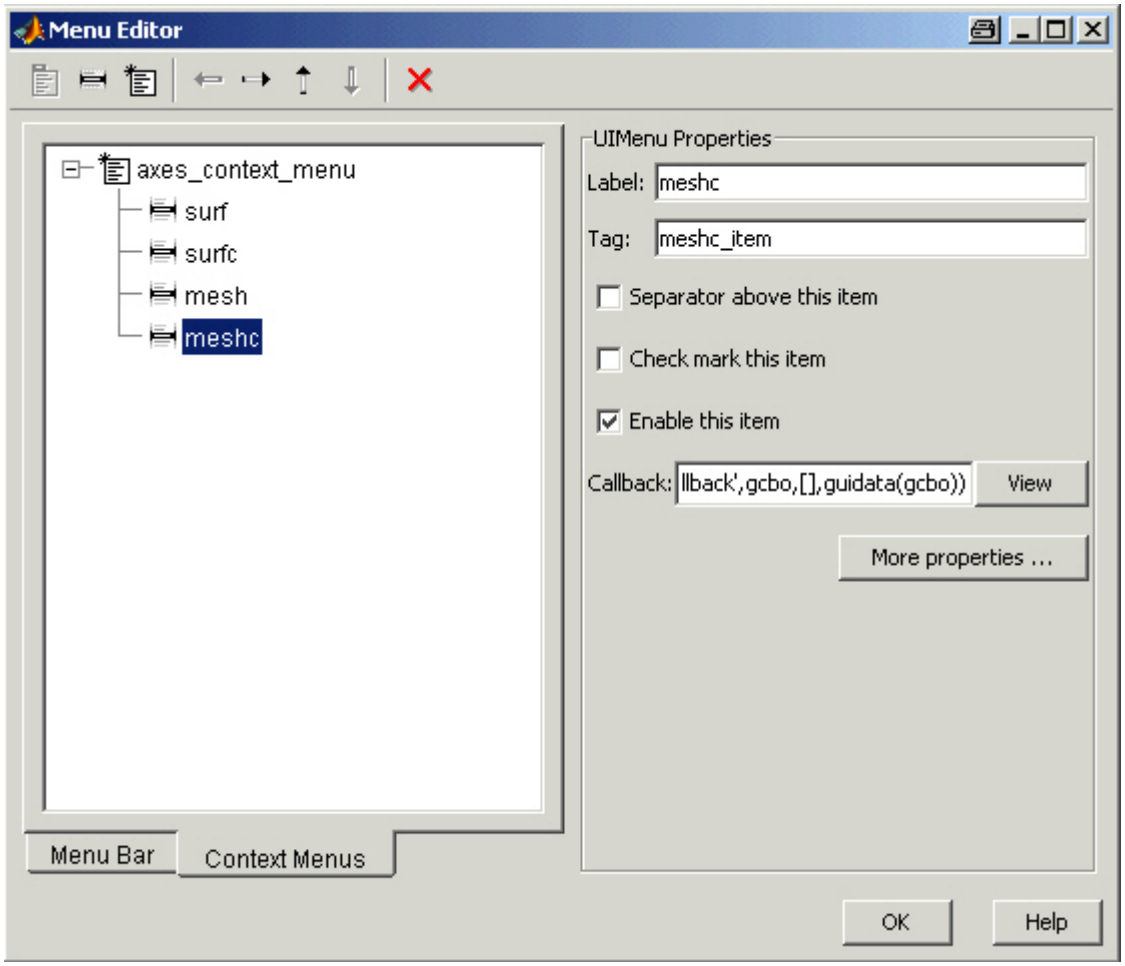
Label: „surf“

Tag: „surf_item“

Es gilt hier aufzupassen, dass beim „Tag“ nicht zweimal derselbe Name für einen Menüeintrag verwendet wird (also z.B. „surf_menu“). Aus den Namen des Tags werden nämlich die Rückruf-Funktionen erzeugt; wird zweimal derselbe Name verwendet, wird eine bereits bestehende Rückruf-Funktion überschrieben (wär hier nicht ganz so kriminell, da ja sowohl in „surf_menu“ als auch „surf_item“ dasselbe passiert; aber das ist ja nicht immer der Fall).

Um ein neuen Eintrag hinzuzufügen, wird wieder „axes_context_menu“ angeklickt und dann der Icon „New Menu Item“.

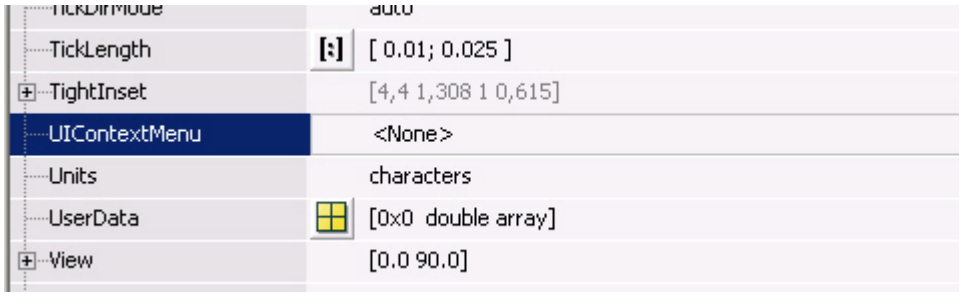
Nach diesem Verfahren werden „surfc“ mit Tag „surfc_item“, „mesh“ mit „mesh_item“ und „meshc“ mit „meshc_item“ als weitere Menüfelder hinzugefügt. Das komplette Kontextmenü hat damit das Aussehen:



Den Editor können wir mittels „OK“ schließen.

Bevor wir die dazugehörigen Rückruf-Funktionen ausfüllen, muß dem System noch mitgeteilt werden, welchem GUI-Element das Kontextmenü zugeordnet werden soll.

In GUIDE klicken wir mit der rechten Maustaste das Graphik-Element „axes1“ an und kommen über das dazugehörige Kontixtmenü (jetzt kennen wir’s ja endlich!) in den Propperty-Inspector.



Dort suchen wir den Eintrag „UIContextMenu“.

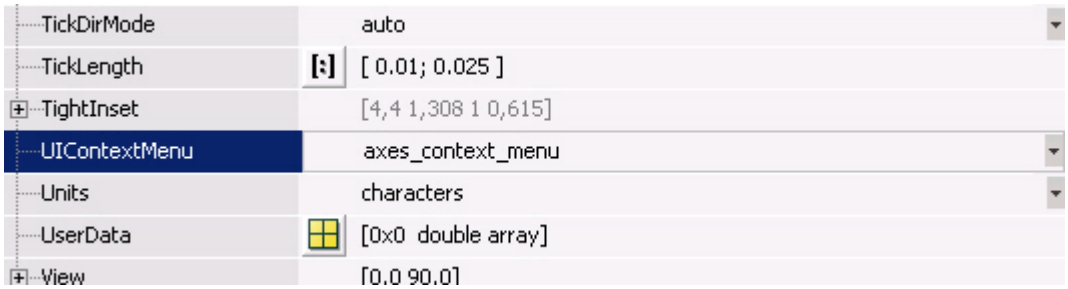
Noch zeigt das Fenster dort den Eintrag „<None>“ an. Anklicken an dieser Stelle bringt jedoch die Auswahl

„<None>“

„axes_contex_menu“

an.

Letzteres ist der Name des Menüs, den wir unter „Tag“ im Menü-Editor eingegeben haben.



Den Eintrag wählen wir nun aus und schließen den Inspector.

Es verbleibt nur noch, die Rückruf-Funktionen

```
function surf_item_Callback(hObject, eventdata, handles)
function surfc_item_Callback(hObject, eventdata, handles)
function mesh_item_Callback(hObject, eventdata, handles)
function meshc_item_Callback(hObject, eventdata, handles)
```

zu implementieren.

Wir übernehmen einfach die Befehlsfolge aus den Rückruf-Funktionen für die Menüfelder der Menüleiste.

Das gesamte Programm hat damit das Aussehen:

```
function varargout = Kap15_15_Gui5_ContextMenu(varargin)
% KAP15_15_GUI5_CONTEXTMENU M-file for Kap15_15_Gui5_ContextMenu.fig
%      KAP15_15_GUI5_CONTEXTMENU, by itself, creates a new
%      KAP15_15_GUI5_CONTEXTMENU or raises the existing singleton*.
%
%      H = KAP15_15_GUI5_CONTEXTMENU returns the handle to a new
%      KAP15_15_GUI5_CONTEXTMENU or the handle to the existing singleton*.
%
%      KAP15_15_GUI5_CONTEXTMENU('CALLBACK',hObject,eventData,handles,...)
%      calls the local function named CALLBACK in
%      KAP15_15_GUI5_CONTEXTMENU.M with the given input arguments.
%
%      KAP15_15_GUI5_CONTEXTMENU('Property','Value',...) creates a new
%      KAP15_15_GUI5_CONTEXTMENU or raises the existing singleton*.
%      Starting from the left, property value pairs are applied to the GUI
%      before Kap15_15_Gui5_ContextMenu_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to
%      Kap15_15_Gui5_ContextMenu_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% Kap15_15_Gui5_ContextMenu
```

```
% Last Modified by GUIDE v2.5 25-Jul-2007 14:19:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_15_Gui5_ContextMenu_OpeningFcn,
...
                   'gui_OutputFcn',  @Kap15_15_Gui5_ContextMenu_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_15_Gui5_ContextMenu is made visible.
function Kap15_15_Gui5_ContextMenu_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_15_Gui5_ContextMenu (see
%            VARARGIN)

% Choose default command line output for Kap15_15_Gui5_ContextMenu
handles.output = hObject;

x = -10 : 0.5 : 10;
y = x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.*xx + yy.*yy);
handles.z = (sin(r) + 0.001)./(r + 0.001);
surf(handles.z);

set(handles.axes1, 'Position', [10 10 70 20]);

set(handles.uipanel1, 'Title', 'Ansicht');
set(handles.uipanel1, 'Position', [83 2 20 10]);

set(handles.radiobutton1, 'String', 'surf');
set(handles.radiobutton1, 'Position', [1 7  18 2]);
set(handles.radiobutton2, 'String', 'surfc');
set(handles.radiobutton2, 'Position', [1 5 18 2]);
set(handles.radiobutton3, 'String', 'mesh');
set(handles.radiobutton3, 'Position', [1 3 18 2]);
set(handles.radiobutton4, 'String', 'mesh');
set(handles.radiobutton4, 'Position', [1 1 18 2]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_15_Gui5_ContextMenu wait for user response (see
% UIRESUME)
```

```
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_15_Gui5_ContextMenu_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function datei_menu_Callback(hObject, eventdata, handles)
% hObject    handle to datei_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function datei_menu_ende_Callback(hObject, eventdata, handles)
% hObject    handle to datei_menu_ende (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
delete(handles.figure1);

% --------------------------------------------------------------------
function ansicht_menu_Callback(hObject, eventdata, handles)
% hObject    handle to ansicht_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function surf_menu_Callback(hObject, eventdata, handles)
% hObject    handle to surf_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
surf(handles.z);
set(handles.radiobutton1, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function surfc_menu_Callback(hObject, eventdata, handles)
% hObject    handle to surfc_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
surfc(handles.z);
set(handles.radiobutton2, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function mesh_menu_Callback(hObject, eventdata, handles)
% hObject    handle to mesh_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mesh(handles.z);
set(handles.radiobutton3, 'Value', 1);
guidata(hObject, handles);
```

```
% --------------------------------------------------------------------
function meshc_menu_Callback(hObject, eventdata, handles)
% hObject    handle to meshc_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
meshc(handles.z);
set(handles.radiobutton4, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function hilfe_menu_Callback(hObject, eventdata, handles)
% hObject    handle to hilfe_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function info_menu_Callback(hObject, eventdata, handles)
% hObject    handle to info_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Gui5_Info;

% --------------------------------------------------------------------
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(hObject,'Tag')   % Get Tag of selected object
    case 'radiobutton1'
        % Code for when radiobutton1 is selected.
        surf(handles.z);
    case 'radiobutton2'
        % Code for when radiobutton2 is selected.
        surfc(handles.z);
    case 'radiobutton3'
        % Code for when togglebutton1 is selected.
        mesh(handles.z);
    case 'radiobutton4'
        % Code for when togglebutton2 is selected.
        meshc(handles.z);
    otherwise
        % Code for when there is no match.
        warning('Fall bei Aufruf Radio Buttons vergessen!');
        fprintf('tag = %s\n', get(h.Object,'Tag'));
end

% --------------------------------------------------------------------
function axes_context_menu_Callback(hObject, eventdata, handles)
% hObject    handle to axes_context_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function surf_item_Callback(hObject, eventdata, handles)
% hObject    handle to surf_item (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
surf(handles.z);
set(handles.radiobutton1, 'Value', 1);
guidata(hObject, handles);
```

```
% --------------------------------------------------------------------
function surfc_item_Callback(hObject, eventdata, handles)
% hObject    handle to surfc_item (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
surfc(handles.z);
set(handles.radiobutton2, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function mesh_item_Callback(hObject, eventdata, handles)
% hObject    handle to mesh_item (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mesh(handles.z);
set(handles.radiobutton3, 'Value', 1);
guidata(hObject, handles);

% --------------------------------------------------------------------
function meshc_item_Callback(hObject, eventdata, handles)
% hObject    handle to meshc_item (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
meshc(handles.z);
set(handles.radiobutton4, 'Value', 1);
guidata(hObject, handles);
```

Klickt man jetzt also im GUI mit der rechten Maustaste auf das Graphik-Element, erscheint ein Kontextmenü mit der Auswahl “surf”, “surfc”, “mesh” und “meshc” – durch Anklicken eines solchen Eintrags wird der entsprechende Radio-Button gesetzt und die dazugehörige Ansicht präsentiert.

Bereits in der Einleitung dieses Kapitels wurden spezielle GUIs vorgestellt, die sogenannten Dialog-Boxen. Diese wollen wir als nächstes betrachten.

Dialog-Boxen

Auf Deutsch gilt die Entsprechung „Dialogfeld“.

Ein solcher GUI besteht aus einem Dialogfenster und einem oder mehreren Dialogelementen, etwas Text oder Listen oder Schaltflächen (push buttons).

Einige Dialog-Boxen sind

msgbox	erzeugt die Ausgabe einer einfachen Meldung
warndlg	erzeugt die Ausgabe einer Warnungsmeldung
errordlg	erzeugt die Ausgabe einer Fehlermeldung
helpdlg	erzeugt die Ausgabe eines Hilfetextes
questdlg	erzeugt eine „Ja-Nein-Abbruch“-Abfragendialog

inputdlg	erzeugt einen Dialog zur Eingabe des Textes
uiputfile	startet einen Dialog zur Auswahl einer Datei zum Speichern
uigetfile	startet einen Dialog zur Auswahl einer Datei zum Lesen
uigetdir	startet einen Dialog zur Auswahl eines Verzeichnisses mit Pfad
printdlg	startet den Drucker-Auswahldialog für akuelle Graphik
uisetcolor	startet den Farbdialog zur Auswahl einer RGB-Farbe

Der Aufruf zum Setzen eines solchen Dialogfeldes kann an einer beliebigen Stelle im Programm erfolgen.

Dialogfelder können im sogenannten modalen oder nicht-modalem Modus gesetzt werden. Modale Felder sind solche, auf die der Anwender in jedem Fall reagieren muß. Das bedeutet, daß er so lange nicht weiterarbeiten kann, bis er das Feld geschlossen hat (siehe auch Beispiel `Kap15_08_Gui4Rueckgabe`). Nicht-modale Fenster hingegen erlauben dem Benutzer, in der Anwendung weiterzuarbeiten, während das Fenster geöffnet bleibt.

Mittels Parameter können die Fenster geeignet gesetzt werden – allerdings nur in gewissem Rahmen, der durch die Parameter vorgegeben ist.

questdlg („Frage-Antwort“)
Der allgemeine Befehl zur Erzeugung lautet:

```
button =  questdlg('qstring','title','str1','str2','str3','default')
```

Dabei ist

`qstring`	beinhaltet Frage
`title`	Überschrift der Dialogbox
`str1, str2, str3`	die drei möglichen Antworten als Aufschrift auf den Schaltflächen
`default`	eine der drei Antworten, die auf aktiviert wird, wenn statt einer Schaltfläche die Entertaste betätigt wird.

Wie zu ersehen, gibt es also bei diesem Typ von Dialogfeldern nur drei mögliche Schaltflächen als Antwort auf die Frage – und nicht mehr.
Rückgabewert ist der String „button“; er enthält den Inhalt von str1, str2 oder str3 je in Abhängigkeit der angeklickten Schaltfläche.
Eine Auswertung kann daher die Gestalt haben:

```
switch button
```

```
    case Inhalt von str1,
      …
    case Inhalt von str2,
      …
    case Inhalt von str3,
      …
  end
```

Beipspiel:

Im Kommandofenster gibt man ein:

```
>> button = questdlg('Was ist Ihre Lieblingsfarbe', 'Farbauswahl', 'rot', 'grün', 'blau', 'blau')
```

Es erscheint auf dem Bildschirm folgende Dialogfläche:

Der Modus ist „modal“, d.h.. um mit MATLAB fortfahren zu können, muß man entweder das Fenster schließen oder eben einen der Knöpfe drücken. Betätigz man die Schaltfläche „grün“, verschwindet das Fenster, und im Kommandofenster erscheint die Meldung

```
button =

grün

>>
```

Eine konkrete Anwendung siehe in den ersten beiden Beispielen zu Beginn dieses Kapitels.

msgbox („Mitteilung“)

Ein Befehl zur Erzeugung lautet:

```
h = msgbox(message,title,'icon', createmode)
```

Dabei ist

`message`	der Inhalt der eigentlichen Mitteilung
`title`	die Überschrift
`icon`	der Icon, der als zusätzliche Info ausgegeben werden kann. Standardwert ist ‚`none`’. Andere Standardicons sind ‚`error`’, ‚`help`’ und ‚`warn`’.
`createmode`	unterscheidet den Modus modal, nicht-modal. Es kann gesetzt werden: ‚`modal`’, ‚`non-modal`’. Standardwert ist ‚`non-modal`’

Rückgabewert `h` ist der Handle auf die Box.

Die Box wird standardmäßig mit einer Schaltfläche „OK“ versehen; wird diese betätigt, verschwindet die Box.

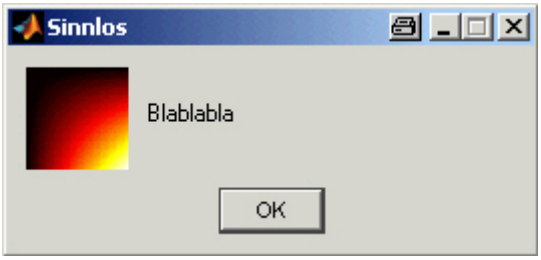
`h = msgbox(Message,Title,'custom',IconData,IconCMap)` ist eine Erweiterung für selbst definierte Icons (da ist – wie zu ersehen – ‘`icon`’ = ‘`custom`’ gesetzt). Einzelheiten bitte der Dokumentation zu MATLAB entnehmen („`help msgbox`“).

Beispiel:

Eingabe im Kommandofenster:

```
>> Data=1:64;Data=(Data'*Data)/64;
>> h=msgbox('Blablabla','Sinnlos','custom',Data,hot(64))
```

Es erscheint (Modus nicht-modal) die Mitteilung



Nach Betätigen des Knopfes „OK“ erscheint im Kommandofenster

```
h =

    0.0027
```

warndlg („Warnhinweis’)

Dies ist im Prinzip eine spezielle Mitteilung.

Ein Befehl zur Erzeugung lautet

```
h = warndlg('warnstring','dlgname',createmode)
```

Dabei ist

`warnstring`	der Inhalt der Warnung
`dialogname`	Überschrift; wenn weggelassen, wird standardmäßig „Warning Dialog“ ausgeben
`createmode`	unterscheidet den Modus modal, nicht-modal. Es kann gesetzt werden: ‚`modal`’, ‚`non-modal`’. Standardwert ist ‚`non-modal`’.

Rückgabewert ist der Handle `h` auf die Box.

Standardmäßig wird das Icon ‚`warn`’ und die Schaltfläche „OK“ gesetzt. Das Feld verschwindet, wenn die Schaltfläche gedrückt wird.

Beispiel:

Eingabe im Kommandofenster die ziemlich sinnlose Warnung:

>> h = warndlg('Drücken von OK löscht Datei-Inhalt','!! Warnung !!')

es erscheint das Fenster

(man wird feststellen, daß dies eine ziemlich sinnlose Warnung ist – welche Möglichkeit hat der Anwender an dieser Stelle, noch etwas zu ändern? Stecker ziehen???)

Es bleibt nichts anderes übrig als „OK“ zu drücken, da der Rückgabewert ja nur der Handle ist:

```
h =

    0.0034
```

errordlg („Fehlermeldung“)

auch dies ist eine spezielle Mitteilung.

Ein Befehl zur Erzeugung lautet

```
h = errordlg('errorstring','dlgname','on')
```

Dabei ist

`errorstring`	Inhalt der Fehlermeldung
`dlgname`	Überschrift
`on`	(optional) – falls Fehlermeldung schon auf dem Bildschirm vorhanden, wird nicht ein zweites Fenster geöffnet, sondern das bereits vorhanden (möglicherweise verdeckte) wieder nach vorn auf den Bildschirm gebracht.

Rückgabewert `h` ist der Handle auf die Box

Standardmäßig ist die Box mit dem Icon ‚`error`‘ und einer Schaltfäche ‚OK’ versehen. Sie verschwindet vom Bildschirm, wenn „OK“ gesetzt oder der Rücksprungschlüssel (das Icon „Kreuz“) betätigt wird.

Beispiel:

Eingabe im Kommandofenster:

```
>> h = errordlg('Dieses Programm läuft schon seit Stunden ohne Fehler. Ich bau jetzt einen Fehler ein', 'Fehler')
```

Es erscheint das Fenster

und nach Betätigen von „OK“ – das Fenster verschwindet im Kommandofenster die Ausgabe des Handles

```
h =

    0.0042
```

helpdlg („Hilfe“)

auch dies ist eine spezielle Mitteilung.

Befehl zur Erzeugung:

```
h = helpdlg('helpstring','dlgname')
```

Dabei ist

`helpstring` der Inhalt der Aussage im Fenster

`dlgname` die Überschrift

der Rückgabewert ist wieder der Handle `h` auf die Box

Standardmäßig ist die Box mit dem Icon ‚`help`’ und der Schaltfläche ‚`OK`’ versehen. Wird diese betätigt oder der Rücksprungschlüssel, verschwindet die Box.

Beispiel:

Eingabe im Kommandofenster

```
h = helpdlg('Hahahahhaha!!', 'Sie wollen Hilfe??')
```

führt zur Ausgabe von

(manchem vergeht dabei das Lachen)

Betätigt man „OK“, verschwindet das Fenster, und im Kommandofenster erscheint

```
h =

    0.0045
```

inputdlg

Dies weicht endlich mal ein bißchen ab.

Die Grundstruktur eines solchen Eingabefensters lautet:

```
answer = inputdlg(prompt)
```

oder etwas erweitert

answer = inputdlg(prompt,dlg_title)

Dabei ist

`prompt` Name des Eingabefeldes (ist ein Cell-Objekt)

`dlg_title` die Überschrift

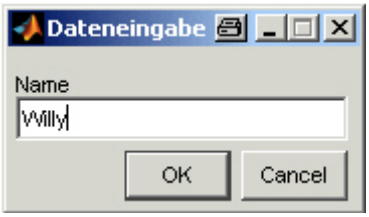
Rückgabewert `answer` ist ein Cell-Objekt, das (in diesem Fall) nur ein Element, nämlich den Eingabestring enthält.

Beispiel:

Eingabe im Kommandofenster

```
answer = inputdlg('Name', 'Dateneingabe')
```

Es erscheint ein Fenster mit einem Eingabefeld. In dieses wird irgendetwas eingetippt:



Betätigen der Schaltfläche „OK“ läßt das Fenster verschwinden – und im Kommandofenster wird ausgegeben:

```
answer =

    'Willy'
```

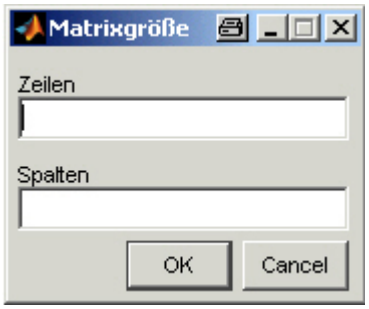
genaugenommen ist answer{1} = „Willy“.

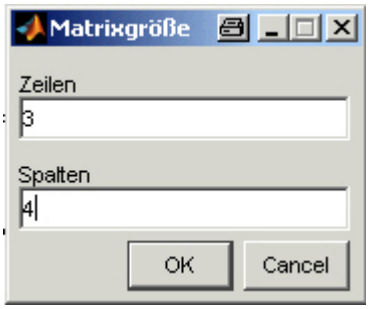
Soll mehr als eine Eingabe in der Box gemacht werden, muß dies im Cell-Objekt prompt angegeben werden.

Beispiel:

```
>> answer = inputdlg({'Zeilen', 'Spalten'}, 'Matrixgröße')
```

bringt eine Dialogfläche mit den zwei Eintragsmöglichkeiten „Zeilen“ und „Spalten“ auf den Bildschirm:





Gibt man nun irgendwelche Werte ein (wie rechts zu ersehen) und schließt mit „OK“, lautet die Ausgabe im Kommandofenster

```
answer =

    '3'
    '4'
```

Genaugenommen ist

answer{1} = ‚3’ und answer{2} = ‚4’ (aufpassen! ‚3’ und ‚4’ sind Zeichenketten, keine Zahlen!)

uiputfile (Dateiauswahl zum Abspeichern)

Zur Auswahl einer Datei zum Abspeichern einer Datei dient das Dialogfeld „uiputfile“.

Ein Befehl zum Erzeugen lautet

```
[FileName,PathName] = uiputfile('FilterSpec','DialogTitle')
```

Dabei ist

`FilterSpec`	legt den Datentyp fest, der angezeigt bzw. abgelegt wird, erkennbar durch die Extension nach dem Punkt des Dateinamens. Dient als sogenannter Filter
`DialogTitle`	ist die Überschrift des Dialogs

Rückgabewert

`FileName`	ist der Name der Datei (einschließlich Extension)
`PathName`	ist der (absolute) Pfad des Verzeichnisses, in dem die Datei abgelegt wird.

Es öffnet sich das bekannte Windows-Fenster zum Abspeichern einer Datei. Mittels eines Browsers kann man sich durch die Verzeichnisse blättern, um einen geeigneten Platz für die

Datei zu finden. Den Dateinamen trägt man dann (in Windows ohne Extension) in dem Texteingabefeld „Dateiname“ ein.

Drückt man „OK“, wird – wenn noch nicht vorhanden – dort eine Datei dieses Namens angelegt (also noch leer) und ihr Name und Pfad als Rückgabewert abgelegt. Das ummantelnde Programm muß dann noch für Sorge tragen, dass wirklich Daten in die Datei geschrieben werden – sonst wird sie nicht angelegt. Ist die Datei bereits vorhanden, erscheint eine Warnungsmeldung, dass die Datei damit überschrieben wird.

Drückt man „Abbrechen“ („Cancel“), wird nichts angelegt – und `FileName` und `PathName` werden auf 0 gesetzt.
In beiden Fällen wird das Fenster geschlossen.

Es gibt noch weitere Varianten – z.B. Setzen der Position des Feldes innerhalb des Bildschirms -, aber dazu sei auf die Dokumentation verwiesen („help uiputfile“).

`uiputfile` zeigt lediglich an, dass eine Datei abgelegt werden kann. Ein Programm muß dann dafür Sorge tragen, dass auch wirklich Daten in die Datei geschrieben werden. Das nachfolgende Programm gibt eine Vorstellung davon.

```
% Kap15_16_Dialogfelder

durchlauf = 'j';
while durchlauf == 'j'
    [fname, pname] = uiputfile('*.txt', ...
        'Datei speichern unter');
    if fname ~= 0
       fid = fopen(fname, 'w');
       if fid ~= -1
          fprintf(fid, 'Hello World');
          fclose(fid);
       else
          h = errordlg(['Konnte Datei ' fname ' nicht anlegen'], ...
            'Fehler', 'modal');
          uiwait(h);
       end;
    else
       h = errordlg('Keine Datei ausgewählt', ...
            'Fehler', 'modal');
       uiwait(h);
    end;
    button = questdlg('noch einen Programmdurchlauf?', 'Abbruch', ...
        'ja', 'nein', 'nein');
    switch button
        case 'ja'
            durchlauf = 'j';
```

```
        case 'nein'
            durchlauf = 'n';
    end;
end;
```

Der entscheidende Befehl lautet also hier

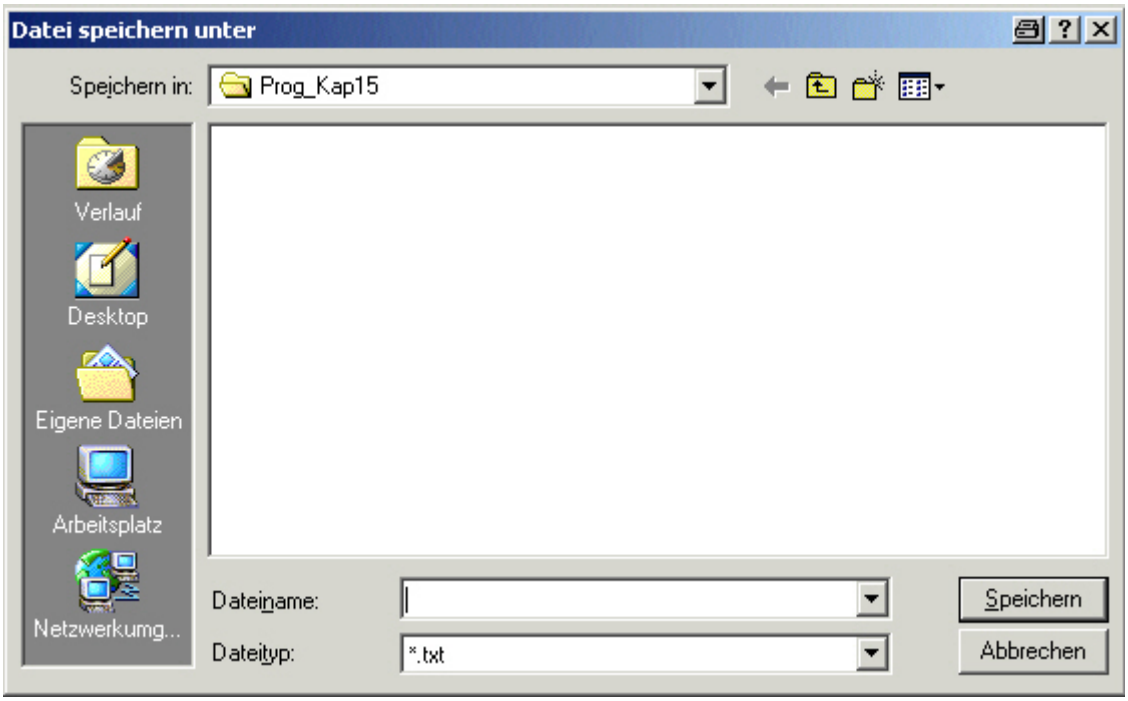
```
[fname, pname] = uiputfile('*.txt',    'Datei speichern unter');
```

Der Filter ist der Dateityp „`*.txt`“ gesetzt. Damit werden alle Dateien mit der Extension „`*.txt`“ angezeigt und, da „`*.*`“ Obermenge davon ist, auch „alle Dateien“.

Der Befehl

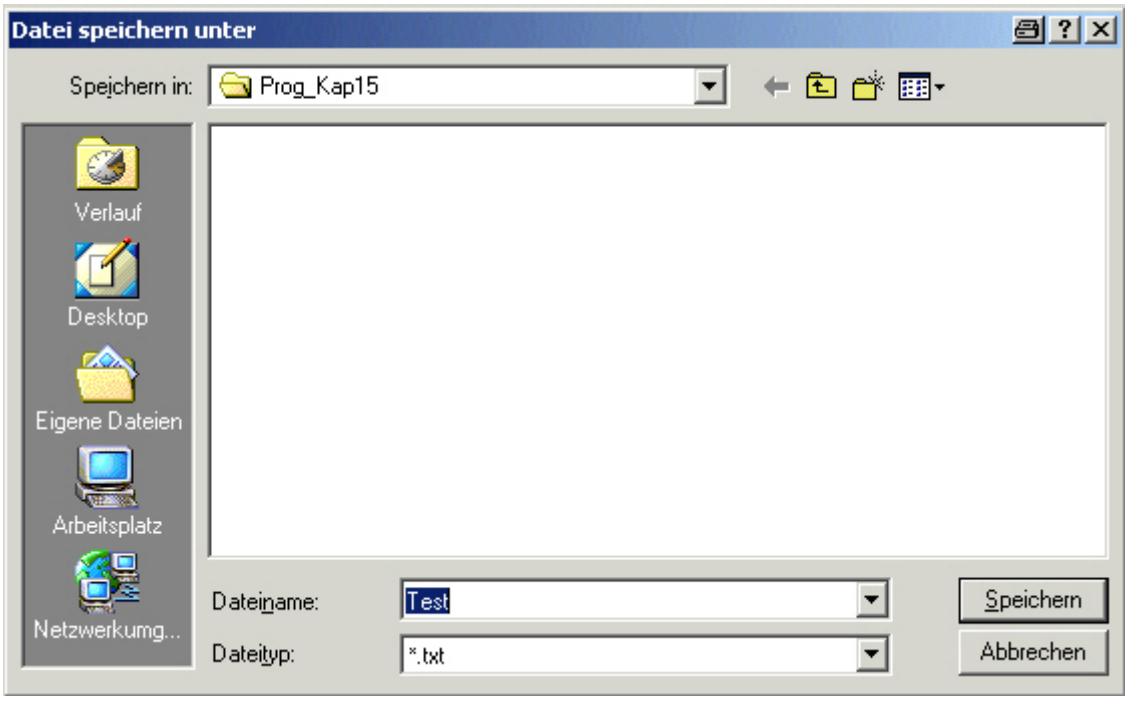
```
>> Kap15_16_Dialogfelder
```

im Kommandofenster bringt also ein Dialogfeld in bekannter Form auf den Bildschirm mit der Aufforderung zur Eingabe eines Dateinamens:



Wie zu ersehen, blinkt der Cursor im Texteingabefeld für Dateiname; Dateityp ist auf „*.txt“ vorbesetzt. Dies ist ein Pop-Up-Menü. Klickt man es an, öffnet sich eine Liste mit einem weiteren zusätzlichen Eintrag: „All Files (*.*)“ – kann man anklicken, damit alle Ordner und Dateien dieses Verzeichnisses (hier „Prog_Kap15“) angezeigt werden können.

Eingabe von Dateiname „Test“ (also ohne Extension!)

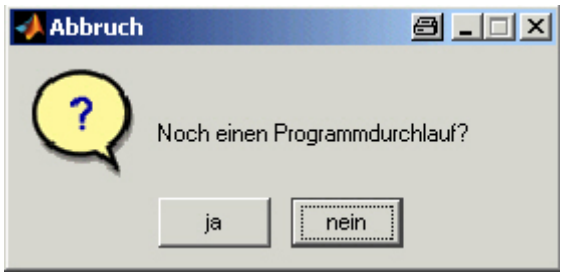


und Betätigen der Schaltfläche „Speichern“ führt dann zum Schließen des Fensters, dem Anlegen einer Datei an dieser Stelle. Das Programm wird fortgesetzt. Insbesondere ist jetzt

`fname` = „Test.txt“

Das Programm öffnet diese Datei, schreibt das langweilige „Hello World“ hinein und schließt sie wieder.

Die Frage auf noch einen Durchlauf ist hier mit einem Frage-Dialog realisiert:



Das Programm reagiert entsprechend darauf: bei Anklicken von „nein“ (was hier als Standard gesetzt ist), wird das Programm beendet.

Falls während der Abfrage auf Eingabe eines Dateinamens der Anwender in “Datei speichern unter“ die Schaltfläche „Abbrechen“ betätigt, ist der Rückgabewert (insbesondere `fname`) gleich 0. Dann wird eine Fehlermeldung ausgegeben:

Erst wenn „OK“ betätigt ist, kann das Programm fortfahren – in diesem Fall mit der Frage auf noch einen Programmdurchlauf.

Für Ausgabe dieser Fehlermeldung ist noch etwas anzumerken:
Es muß dafür Sorge getragen werden, dass das Programm anhält, weil es sonst brutal diese Dialogffäche ausgibt, trotz Modus „modal“ munter weitermacht und das nächste Dialogfeld, nämlich die Frage auf noch einen Programmdurchlauf, auf den Bildschirm bringt und auch noch tückischerweise genau über die Fehlermeldung legt.

Dann allerdings hält es an, erwartet aber, dass sich der Anwender erst einmal um die oben liegende Schaltfläche kümmert.

Dies ist nicht so ganz im Sinne des Erfinders.
Die Reihenfolge sollte die sein:
Die Fehlermeldung wird ausgegeben.
Programm wartet, bis Anwender „OK“ drückt.
Programm fährt fort und bringt nächstes Dialogfeld auf den Bildschirm.

Das Warten des Programms wird durch die Befehlsfolge

```
h = errordlg('Keine Datei ausgewählt', 'Fehler', 'modal');
uiwait(h);
```

realisiert.
Wie (vielleicht noch) erinnerlich, stoppt `uiwait` das Programm, bis der GUI geschlossen wird oder der Befehl `uiresume` erfolgt. Doch welcher GUI soll geschlossen werden? Offensichtlich die Dialogfläche der Fehlermeldung. Auf diese kann mittels des Handles dazugegriffen werden. Daher wird bei uiwait auch noch der Parameter h übergeben (ohnne diesen, also nur durch den Aufruf `uiwait`, macht das Programm nur Unsinn, indem es einen leeren GUI unter figure1 auf den Bildschirm bringt, um anschließend zusätzlich mit der Dialogschaltfläche auf Programmende zu erfreuen. Man probiere es einmal aus!).

uisetcolor (Farbauswahl)

diese Dialogfläche dient der Farbauswahl.

Ein Befehl zur Erzeugung lautet:

c = uisetcolor

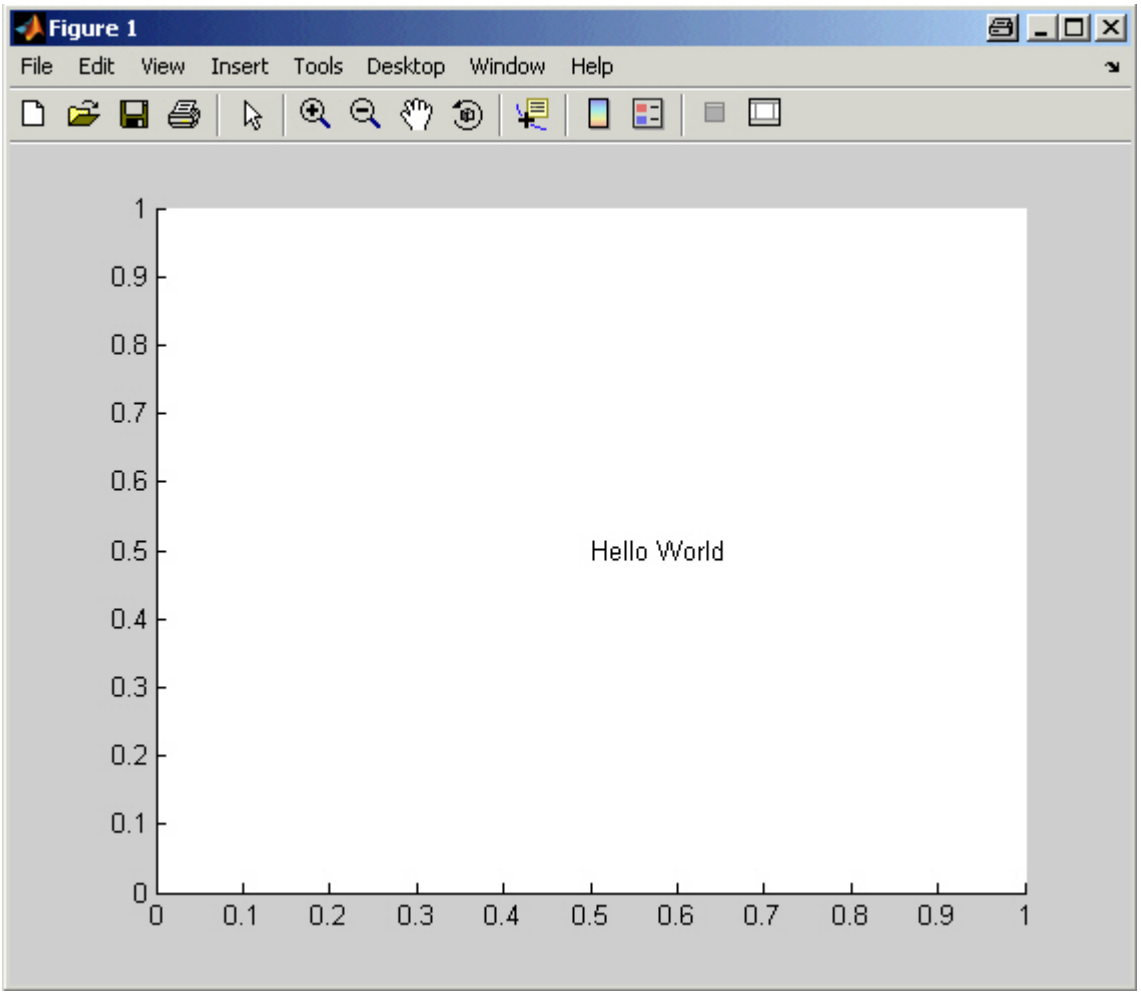
Dann wird eine Farbpalette ausgegeben. Das angewählte Farbfeld wird in seine Rot-Grün-Blau-Anteile zuerlegt dem Vektor c zurückgegeben.

Es können auch Parameter übergeben werden. Dies zeigt am besten das nachfolgende Beispiel:

Im Kommandofenster wird eingegeben:

```
>> hText = text(.5,.5,'Hello World');
```

es erscheint folgendes Fenster:



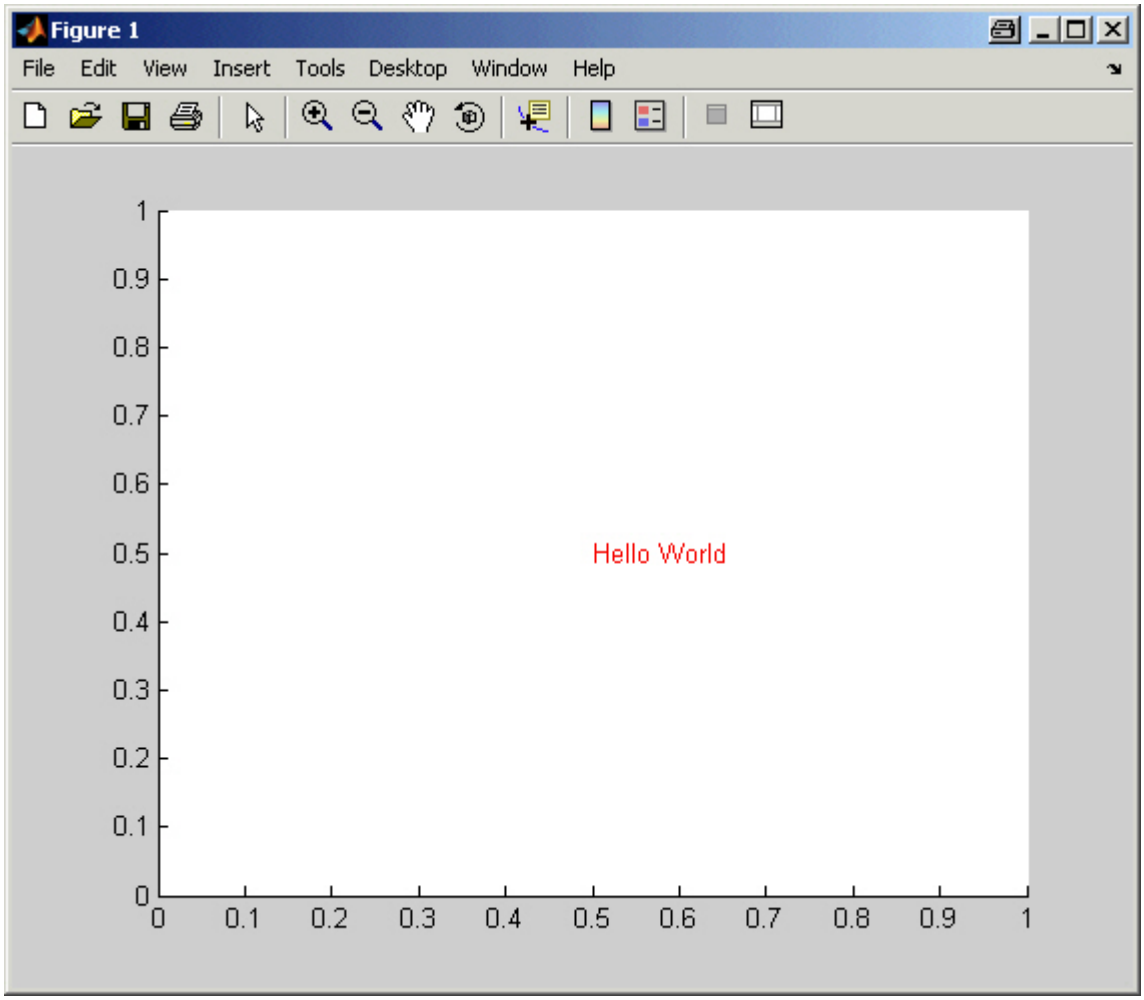
es ist alse ein GUI mit einem Graphik-Feld geöffnet worden, in das ab Position x = 0,5, y = 0,5 beginnend der Text „Hello World“ geschrieben wurde. Zu diesem GUI wurde der Handle `hText` angelegt.

Jetzt wird ins Kommandofenster eingegeben:

```
>> C = uisetcolor(hText, ' bitte Farbe auswählen')
```

es erscheint folgende Dialogfläche:

Man klickt eine Farbe an – sagen wir das erste Kästchen in der zweiten Reihe (also rot) und dann „OK“. Die Dialogfläche verschwindet, der Graphik-Gui hat jetzt das Aussehen:



und im Kommandofenster wird ausgegeben:

```
C =

     1     0     0

>>
```

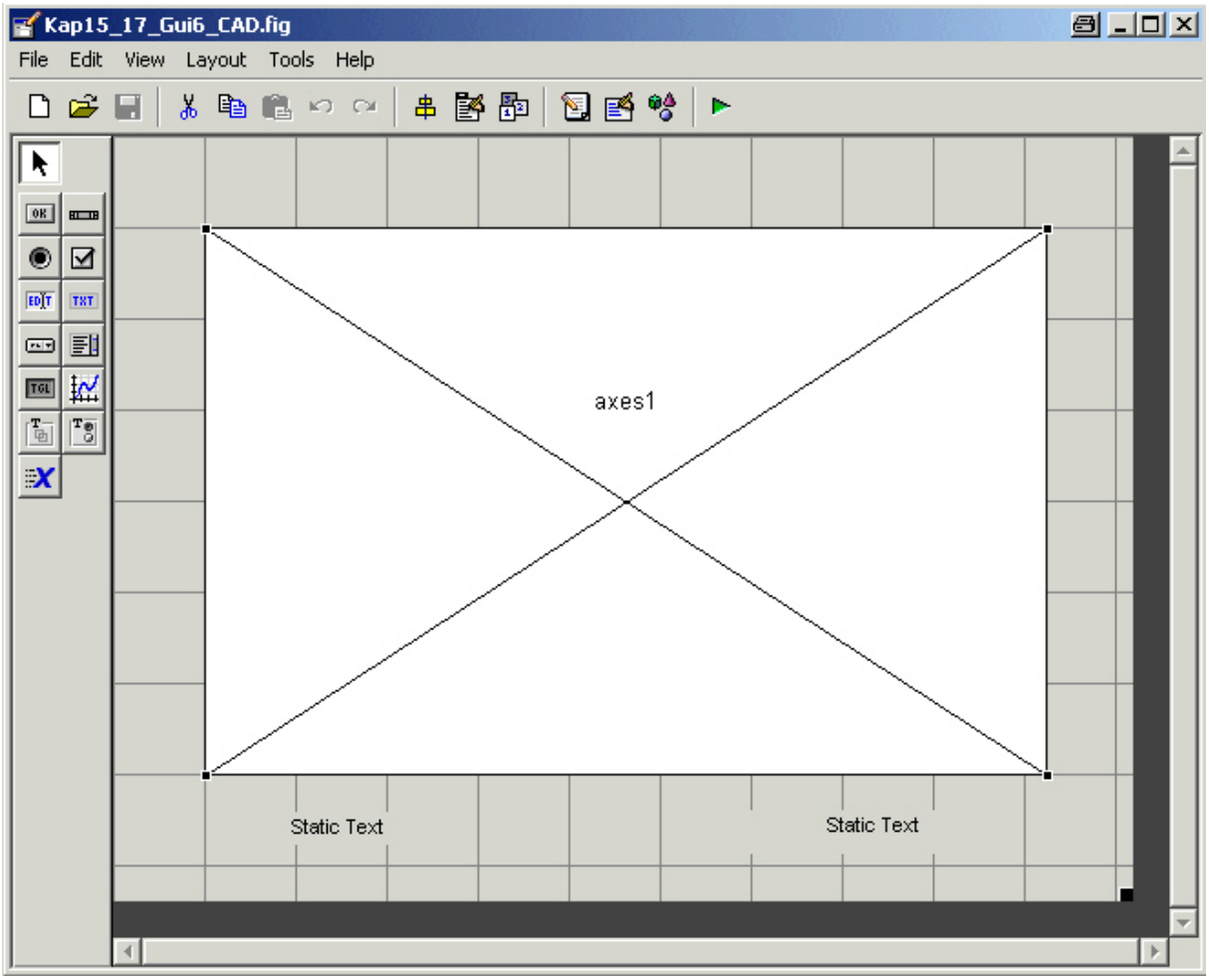
Weitere Details zu Dialogflächen studiert man am besten in der Dokumentation, wenn spezielle Programmiererfordernisse eine geeignete Ein-/Ausgabe notwendig machen.

Maus- Interaktionen

Sämtliche Aktionen eines GUI's wie Auswahl in einer Liste oder Betätigen einer Schaltfläche werden mit der Maus gesteuert. Eine ganz spezielle Interaktion mit der Maus lernten wir bei Kontext-Menüs kennen: mit rechtem Mausklick auf ein selektiertes GUI-Objekt öffnete – insofern implementiert – ein weiteres Menü.

Nachfolgend wollen wir selber bestimmen, welche Aktion bei einem Mausklick erfolgen soll. Als Beispiel wählen wir ein (reduziertes) zweidimensionales Zeichenprogramm, das zwei ausgewählte Punkte durch eine Linie verbinden soll. Dazu erstellen wir innerhalb eines GUI's ein Graphikelement, in dem die Linie gezeichnet werden soll. Die Punkte werden nicht manuell durch Koordinaten über ein Textfeld eingegeben, sondern durch Anklicken mit der (linken) Maustaste innerhalb der Zeichenfläche bestimmt (sogenanntes „Picken").

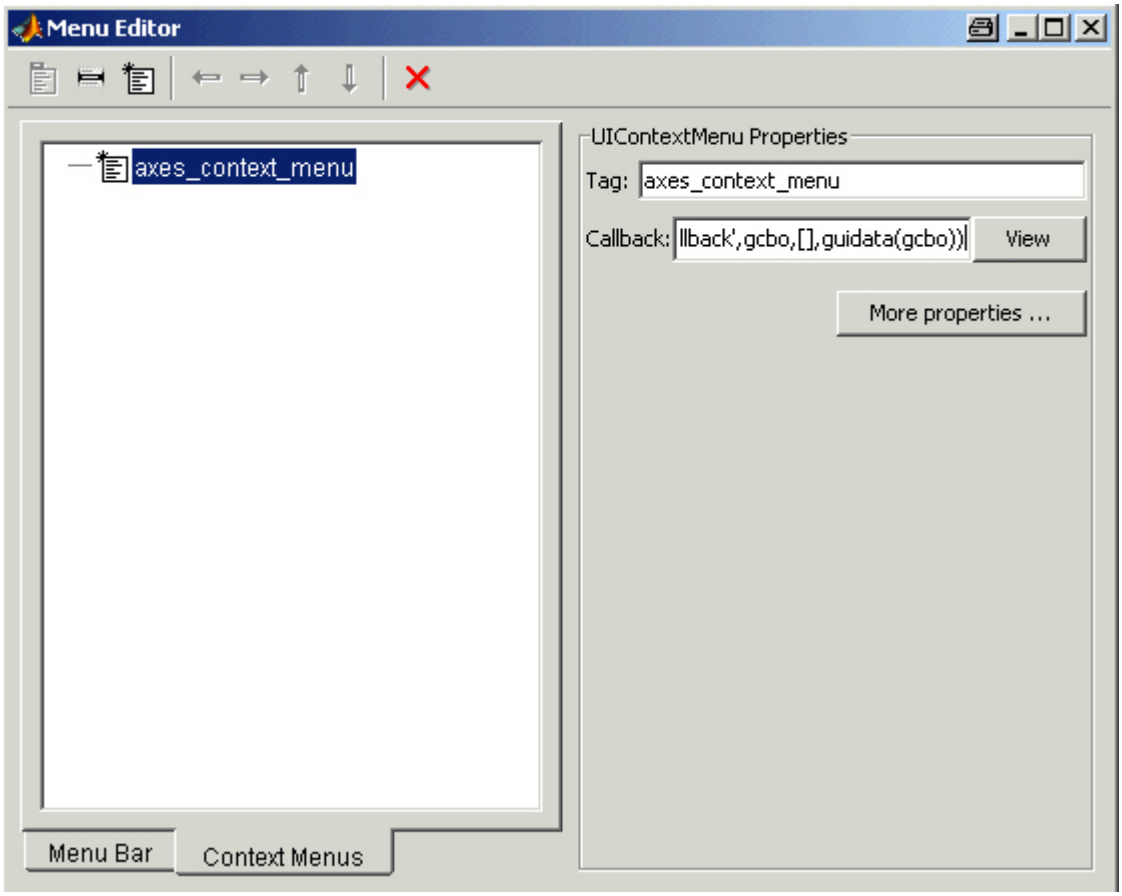
Wir erstellen einen neuen GUI in GUIDE:
In diesem legen wir ein Zeichenelement und zwei (statische) Textelemente ab und speichern alles in „Kap15_17_Gui6_CAD".



Das Zeichnen soll über ein Kontext-Menü gestartet werden. Dazu öffnen wir in GUIDE also den Menü-Editor („Tools" -> „Menu Editor ...."). Dort – wie im letzten Abschnitt beschrieben – erzeugen wir ein ein Kontext-Menü unter dem Einrag „axes_context_menu":

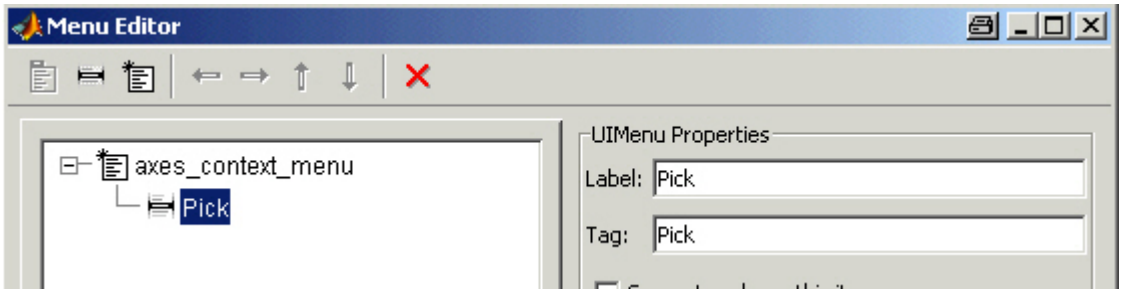
- Tabreiter „Context Menu"
- Icon „New Context Menu"

- in Auswahlliste „Untitled 1“ anklicken
- rechtes obere Feld „Tag“ ausfüllen – der linke Eintrag wird automatisch angepasst

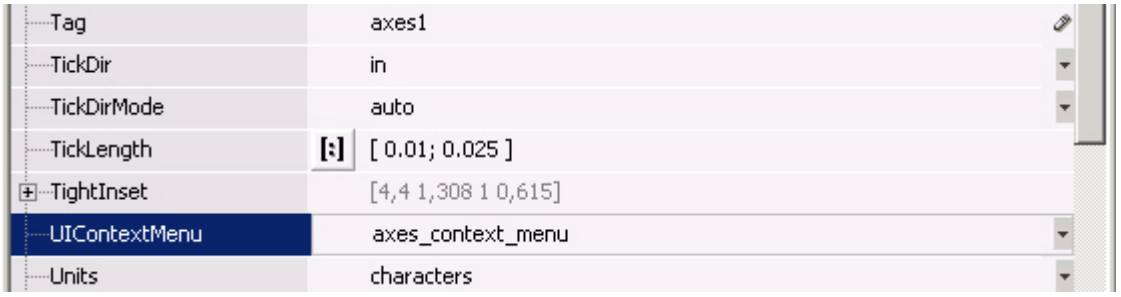


Dann wird das Menüfeld „Pick“ eingetragen:

- anklicekn „axes_context_menu“
- Icon „New Item“
- “Untitled2” anklicken
- Label: “Pick”
- Tag: “Pick”



zu guter Letzt muß noch über den Property-Inspector dieses Kontext-Menü mit dem Graphik-Element „axes1“ verknüpft werden:



Im Eintrag „UIContextMenu“ wird der Eintrag „axes_context_menu“ ausgewählt.

Damit sind die graphischen Vorarbeiten abgeschlossen.
Als nächstes sind die eigentlichen Aktionen im dazugehörigen m-File zu implementieren.

Es befindet sich u.a. im File die Rückrufaktion „`Pick_Callback(hObject, eventdata, handles)`“. Diese wird aufgerufen, wenn das Graphik-Objekt mit der rechten Maustaste selektiert und im dann erscheinenden Kontext-Menü das Feld „Pick“ angeklickt wird. Die geplante Aktion besteht dann darin, mit der linken Maustaste einen Punkt im Graphik-Objekt anzuklicken und dessen Koordinaten zurückzugeben.
Der dazugehörige Befehl lautet `ginput`:

```
[x,y] = ginput(n)
```

Dieser Befehl gibt n x- und y-Koordinaten nach Anklicken in einem graphischen Objekt zurück.
Bauen wir damit zunächst „`Pick_Callback(hObject, eventdata, handles)`“ auf:

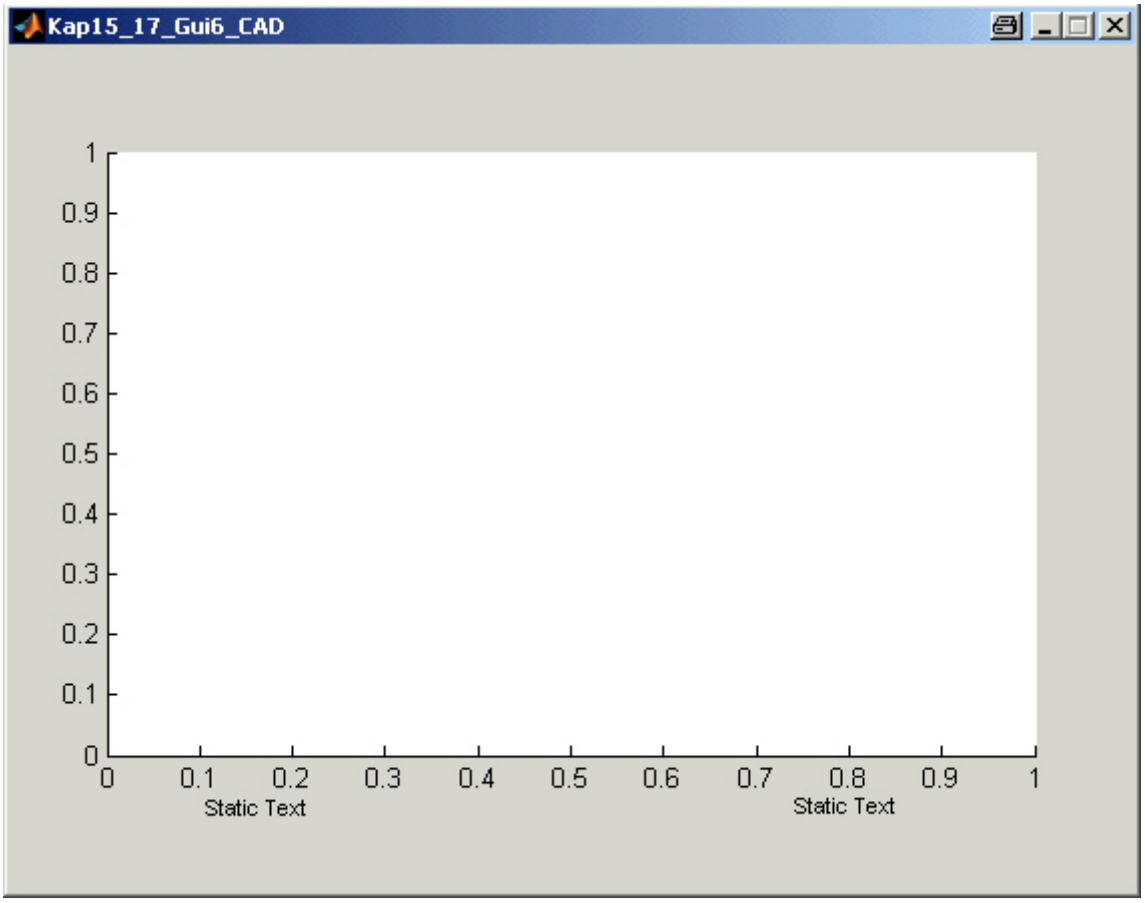
```
function Pick_Callback(hObject, eventdata, handles)
% hObject    handle to Pick (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[x1, y1] = ginput(1);

set(handles.text1, 'String', x1);
set(handles.text2, 'String', y1);
```

Die Funktion ist damit noch nicht komplett – es fehlt ja die Eingabe des zweiten Punktes -, aber es ist schon funktionsfähig.

Aufruf

```
>> Kap15_17_Gui6_CAD
```

im Kommandofenster führt dazu, dass der GUI auf den Bildschirm gebracht wird:



Bewegt man die Maus in das Innere des Graphik-Elementes und klickt die rechte Maustaste, wird das nur aus einem Eintrag bestehende Kontext-Menü geöffnet. Anklicken von „Pick“ führt dann dazu, daß innerhalb des Objekts zwei sich schneidende Geraden erscheinen, die der Mausbewegung folgen. Ein Klick mit der linken Maustaste friert dann die aktuelle Position ein. Diese wird in den unteren beiden Textfeldern ausgegeben. Ansonsten ist allerdings nichts weiter zu sehen – der ausgewählte Punkt selbst bleibt unsichtbar.

Das Programm muß daher noch ein wenig vervollständigt werden:

Es soll noch ein zweiter Punkt ausgewählt werden können; beide Punkte werden dann mit einer Linie verbunden.

Das vollständige Programm hat dann das Aussehen:

```
function varargout = Kap15_17_Gui6_CAD(varargin)
% KAP15_17_GUI6_CAD M-file for Kap15_17_Gui6_CAD.fig
%      KAP15_17_GUI6_CAD, by itself, creates a new KAP15_17_GUI6_CAD or
%      raises the existing singleton*.
%
%      H = KAP15_17_GUI6_CAD returns the handle to a new KAP15_17_GUI6_CAD
%      or the handle to the existing singleton*.
%
%      KAP15_17_GUI6_CAD('CALLBACK',hObject,eventData,handles,...) calls
%      the local function named CALLBACK in KAP15_17_GUI6_CAD.M with the
%      given input arguments.
%
```

```
%      KAP15_17_GUI6_CAD('Property','Value',...) creates a new
%      KAP15_17_GUI6_CAD or raises the existing singleton*.  Starting from
%      the left, property value pairs are applied to the GUI before
%      Kap15_17_Gui6_CAD_OpeningFunction gets called.  An unrecognized
%      property name or invalid value makes property application
%      stop.  All inputs are passed to Kap15_17_Gui6_CAD_OpeningFcn via
%      varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_17_Gui6_CAD

% Last Modified by GUIDE v2.5 30-Jul-2007 12:46:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_17_Gui6_CAD_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_17_Gui6_CAD_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_17_Gui6_CAD is made visible.
function Kap15_17_Gui6_CAD_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_17_Gui6_CAD (see VARARGIN)

% Choose default command line output for Kap15_17_Gui6_CAD
handles.output = hObject;

set(handles.axes1, 'XLim', [0 10]);
set(handles.axes1, 'YLim', [0 10]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_17_Gui6_CAD wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_17_Gui6_CAD_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function axes_context_menu_Callback(hObject, eventdata, handles)
% hObject    handle to axes_context_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Pick_Callback(hObject, eventdata, handles)
% hObject    handle to Pick (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% ersten Punkt auswählen
[x1, y1] = ginput(1);

set(handles.text1, 'String', x1);
set(handles.text2, 'String', y1);

% zweiten Punkt auswählen
[x2, y2] = ginput(1);

x = [x1, x2];
y = [y1, y2];
plot(x, y);

axis([0 10 0 10]);
```

Der `axis`-Befehl zum Schluß stellt sicher, dass die Ausgabe der Linie immer in denselben Dimensionen erfolgt, da MATLAB sonst immer die Fenstergröße automatisch anpassen würde. Daher wurden auch in der Öffnungsfunktion „`Kap15_17_Gui6_CAD_OpeningFcn`“ die Abmessungen für das Graphik-Element gesetzt:

```
set(handles.axes1, 'XLim', [0 10]);
set(handles.axes1, 'YLim', [0 10]);
```

Einen Schwachpunkt hat das Graphik-Programm allerdings noch:
Es kann nur eine Linie gezeichnet werden. Nachdem nämlich dies geschehen ist, ist die Rückruf-Funktion abgearbeitet. Ein neuer rechter Mausklick zum Zeichnen einer weiteren Linie speichert gibt erst erneut die Möglichkeit, eine weitere Linie zu zeichnen – in ein völlig neues Fenster. Man müsste also die bereits ausgewählten Punkte in handles speichern und vorher zeichnen lassen.

Dies realisiert das nachfolgende (vervollständigte) Programm, das in „Kap15_18_Gui6_CAD“ abgelegt ist (es handelt sich dabei um eine Kopie des vorangegangenen „Kap15_17_Gui6_CAD“ mit Erweiterungen). Es lautet wie folgt:

```
function varargout = Kap15_18_Gui6_CAD(varargin)
% KAP15_18_GUI6_CAD M-file for Kap15_18_Gui6_CAD.fig
%      KAP15_18_GUI6_CAD, by itself, creates a new KAP15_18_GUI6_CAD or
%      raises the existing singleton*.
%
%      H = KAP15_18_GUI6_CAD returns the handle to a new KAP15_18_GUI6_CAD
%      or the handle to the existing singleton*.
%
%      KAP15_18_GUI6_CAD('CALLBACK',hObject,eventData,handles,...) calls
%      the local function named CALLBACK in KAP15_18_GUI6_CAD.M with the
%      given input arguments.
%
%      KAP15_18_GUI6_CAD('Property','Value',...) creates a new
%      KAP15_18_GUI6_CAD or raises the existing singleton*.  Starting from
%      the left, property value pairs are applied to the GUI before
%      Kap15_18_Gui6_CAD_OpeningFunction gets called.  An unrecognized
%      property name or invalid value makes property application stop.  All
%      inputs are passed to Kap15_18_Gui6_CAD_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
%      Variation von vorangegangener Version;
%      bereits gezeichnete Punkte werden zwischengespeichert
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_18_Gui6_CAD

% Last Modified by GUIDE v2.5 30-Jul-2007 12:46:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_18_Gui6_CAD_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_18_Gui6_CAD_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_18_Gui6_CAD is made visible.
function Kap15_18_Gui6_CAD_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_18_Gui6_CAD (see VARARGIN)

% Choose default command line output for Kap15_18_Gui6_CAD
handles.output = hObject;

set(handles.axes1, 'XLim', [0 10]);
set(handles.axes1, 'YLim', [0 10]);

handles.start = 1;
handles.vektor = zeros(2, 2);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_18_Gui6_CAD wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kap15_18_Gui6_CAD_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function axes_context_menu_Callback(hObject, eventdata, handles)
% hObject    handle to axes_context_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Pick_Callback(hObject, eventdata, handles)
% hObject    handle to Pick (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% ersten Punkt auswählen
[x1, y1] = ginput(1);

set(handles.text1, 'String', x1);
set(handles.text2, 'String', y1);

% zweiten Punkt auswählen
[x2, y2] = ginput(1);

[row col] = size(handles.vektor);

if handles.start == 1
   % am Anfang ist handles.vektor noch leer
   handles.vektor(1, 1) = x1;
   handles.vektor(1, 2) = y1;
   handles.vektor(2, 1) = x2;
   handles.vektor(2, 2) = y2;
   handles.start = 0;
else
```

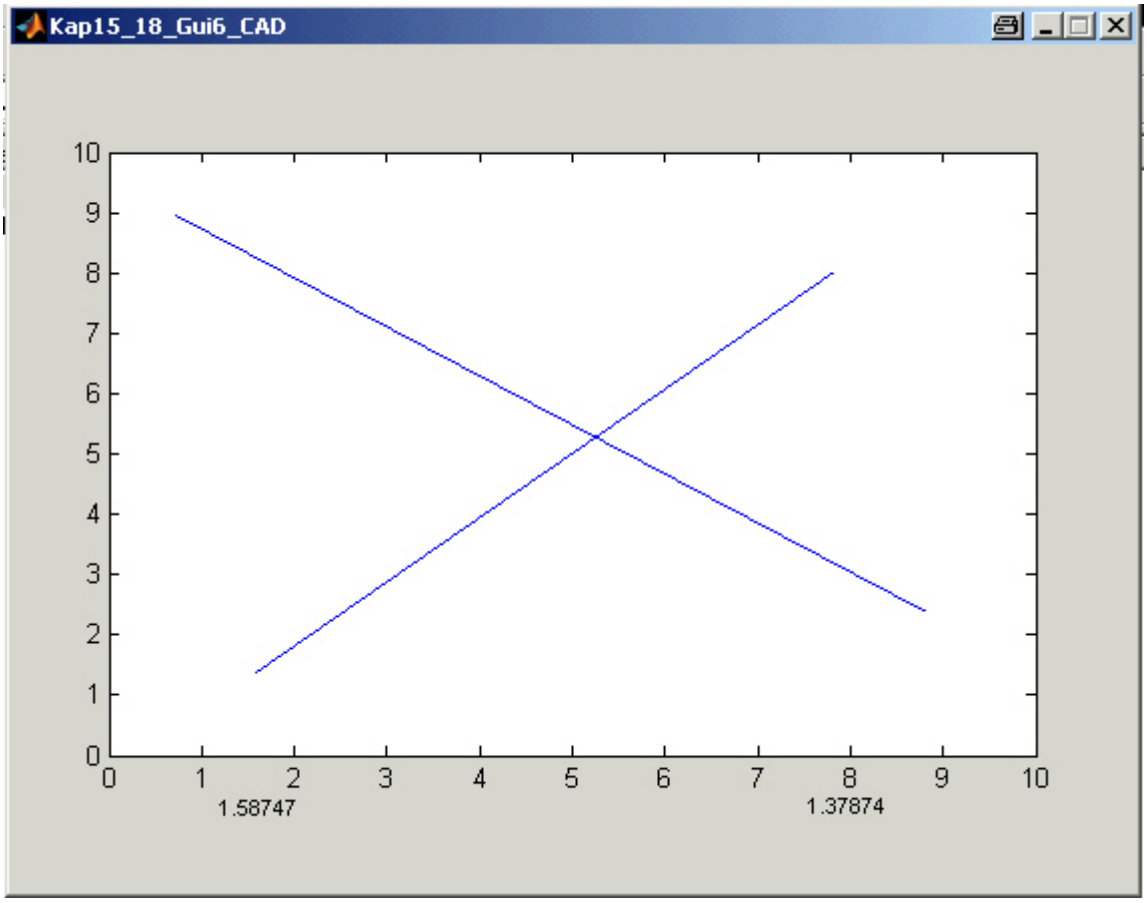
```
   % handles.vektor um zwei Punkte erweitern
   vektor = handles.vektor;
   handles.vektor = zeros(row + 2, 2);
   for i = 1:row
      handles.vektor(i, 1) = vektor(i, 1);
      handles.vektor(i, 2) = vektor(i, 2);
   end;
   handles.vektor(row + 1, 1) = x1;
   handles.vektor(row + 1, 2) = y1;
   handles.vektor(row + 2, 1) = x2;
   handles.vektor(row + 2, 2) = y2;
end;

% ersten Punkt ausgeben - danach hold-Befehl
x = [handles.vektor(1, 1), handles.vektor(2, 1)];
y = [handles.vektor(1, 2), handles.vektor(2, 2)];
plot(x, y);
hold;

% weitere Punke paarweise durch Linien verbinden - falls vorhanden
[zeile spalte] = size(handles.vektor);

i = 3;
while (i <= zeile)
   x = [handles.vektor(i, 1), handles.vektor(i + 1, 1)];
   y = [handles.vektor(i, 2), handles.vektor(i + 1, 2)];
   plot(x, y);
   i = i + 2;
end;

axis([0 10 0 10]);
hold off;

guidata(hObject, handles);
```

Die Erweiterungen sind fett markiert.

Die Idee ist also, daß bereits gezeichnete Punkte in `handles.vektor` gespeichert werden. Dieser wird also in `Kap15_18_Gui6_CAD_OpeningFcn` mit Nullen vorbesetzt, d.h. mit den beiden Punkten (0, 0) und (0, 0).

In `Pick_Callback` werden dann die gespeicherten Punkte um die neuen erweitert und paarweise mit einer Verbindungslinie versehen.

Damit lässt sich nun ein ziemliches Gekrakel verantstalten, wie das nachfolgende Beispiel zeigt:



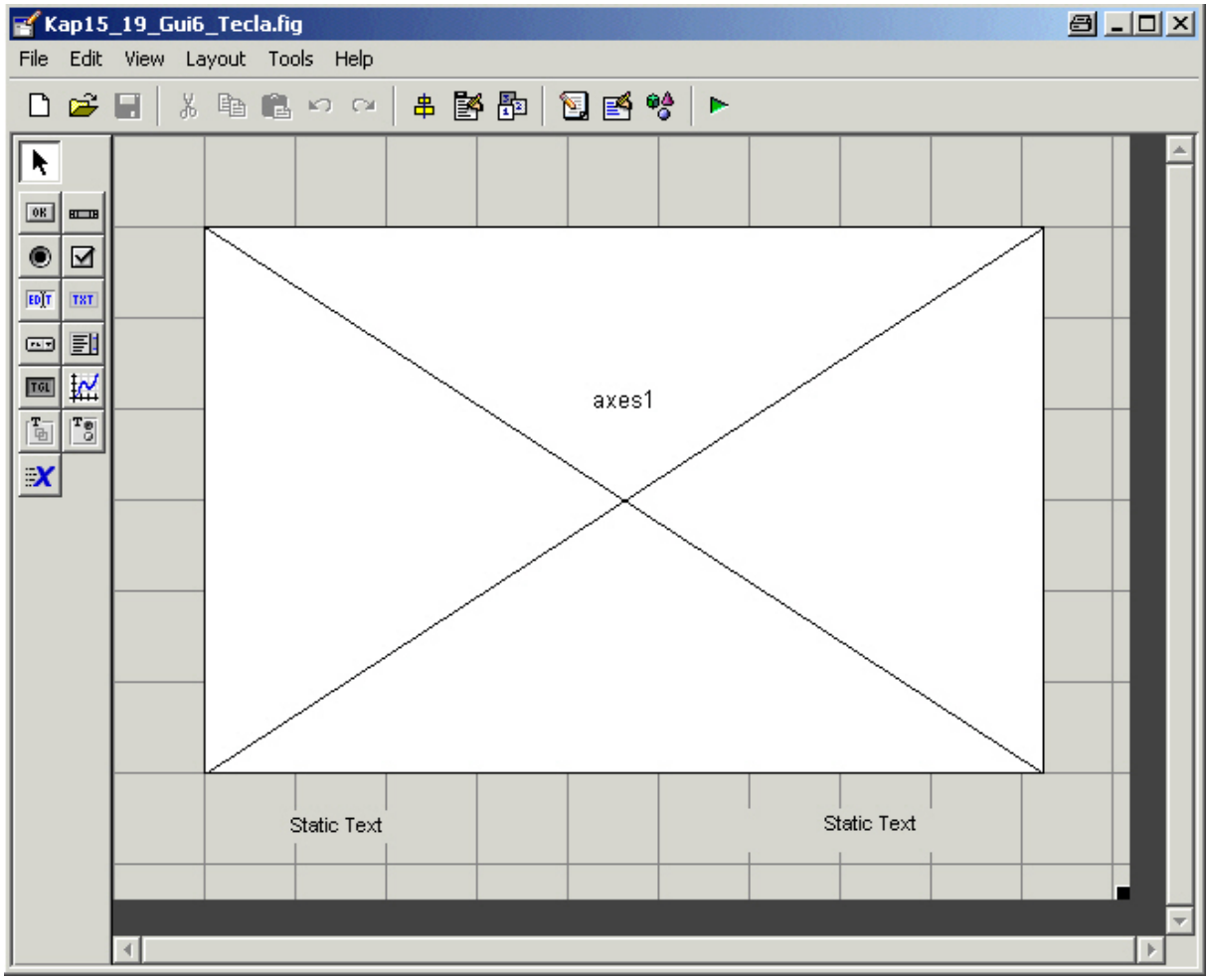
Natürlich ist dies ein sehr rudimentäres Zeichenprogramm – es gibt beispielsweise keine Möglichkeit, Fehler zu korrigieren, Linien zu löschen usw. Aber es zeigt immerhin die Möglichkeiten auf, die mit einem Mausklick in Verbindung stehen.

Tastatur-Interaktionen

Ebenso wie mit einem Mausklick Aktionen ausgelöst werden können, ist dies mit dem Drücken einer ausgewählten Taste möglich.

Zu diesem Zweck muß im Programm noch zusätzlich eine Rückruf-Funktion für Betätigen der Tastatur innerhalb eines GUI's eingefügt werden, eine sogenannte „KeyPress-Function". Eine solche werden wir in das rudimentäre Zeichenprogramm einfügen: drückt der Anwender die Taste „p", sollen zwei Punkte ausgewählt werden und durch eine Linie verbunden werden.

Dazu erstellen wir eine Kopie des vorletzten Programms (das also nur eine einfache Linie zeichnen kann) und speichern diese unter „Kap15_19_Gui6_Tecla.fig" (d.h. GUIDE öffnen, bereits vorhandenen GUI „Kap15_17_Gui6_CAD" öffnen und unter obigem Namen speichern).



In der Menüzeile wählt man den Eintrag „View“ aus, und darin das Feld „View Callbacks“ und in dem sich öffnenden Kontextmenü „KeyPressFcn“.

In der Datei „Kap15_19_Gui6_Tecla.m“ wird eine neue Funktion „`figure1_KeyPressFcn (hObject, eventdata, handles)`“ hinzugefügt. Diese reagiert auf das Drücken einer beliebigen Taste.

Den Wert einer gedrückten Taste kann man mit dem berühmten `get`-Befehl herausholen:

```
c = get(hObject, ,CurrentCharacter');
```

In Abhängigkeit der gedrückten Taste kann man nun irgendwelche Aktionen starten.
Das Programm haben wir wie folgt gestaltet:

```
function varargout = Kap15_19_Gui6_Tecla(varargin)
% KAP15_19_GUI6_TECLA M-file for Kap15_19_Gui6_Tecla.fig
%      KAP15_19_GUI6_TECLA, by itself, creates a new KAP15_19_GUI6_TECLA or
%      raises the existing singleton*.
%
%      H = KAP15_19_GUI6_TECLA returns the handle to a new
```

```
%      KAP15_19_GUI6_TECLA or the handle to the existing singleton*.
%
%      KAP15_19_GUI6_TECLA('CALLBACK',hObject,eventData,handles,...) calls
%      the local function named CALLBACK in KAP15_19_GUI6_TECLA.M with the
%      given input arguments.
%
%      KAP15_19_GUI6_TECLA('Property','Value',...) creates a new
%      KAP15_19_GUI6_TECLA or raises the existing singleton*.  Starting
%      from the left, property value pairs are applied to the GUI before
%      Kap15_19_Gui6_Tecla_OpeningFunction gets called.  An unrecognized
%      property name or invalid value makes property application stop.  All
%      inputs are passed to Kap15_19_Gui6_Tecla_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kap15_19_Gui6_Tecla

% Last Modified by GUIDE v2.5 01-Aug-2007 11:11:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kap15_19_Gui6_Tecla_OpeningFcn, ...
                   'gui_OutputFcn',  @Kap15_19_Gui6_Tecla_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kap15_19_Gui6_Tecla is made visible.
function Kap15_19_Gui6_Tecla_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kap15_19_Gui6_Tecla (see VARARGIN)

% Choose default command line output for Kap15_19_Gui6_Tecla
handles.output = hObject;

set(handles.axes1, 'XLim', [0 10]);
set(handles.axes1, 'YLim', [0 10]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kap15_19_Gui6_Tecla wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = Kap15_19_Gui6_Tecla_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function axes_context_menu_Callback(hObject, eventdata, handles)
% hObject    handle to axes_context_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Pick_Callback(hObject, eventdata, handles)
% hObject    handle to Pick (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% ersten Punkt auswählen
[x1, y1] = ginput(1);

set(handles.text1, 'String', x1);
set(handles.text2, 'String', y1);

% zweiten Punkt auswählen
[x2, y2] = ginput(1);

x = [x1, x2];
y = [y1, y2];
plot(x, y);

axis([0 10 0 10]);



% --- Executes on key press over figure1 with no controls selected.
function figure1_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

c = get(hObject, 'CurrentCharacter');
fprintf('c = %s und im ASCII-Code: %i\n', c, c);
if (c == 'p')
    axis([0 10 0 10]);

    [x1, y1] = ginput(1);
    [x2, y2] = ginput(1);

    x = [x1, x2];
    y = [y1, y2];

    plot(x, y);

    axis([0 10 0 10]);
```

```
end
```

Die Funktion „figure1_KeyPressFcn“ wird bei Drücken jeder Taste aufgerufen, doch nur wenn die Taste „p“ gedrückt wurde, erfolgt eine Aktion – eben das Verbinden zweier selektierter Punkte durch eine Linie.

Falls für die Tastatur-Steuerung nicht die mit Buchstaben belegten, sondern die Sondertasten verwendet werden, empfiehlt es sich, den dazugehörigen ASCII-Wert zu verwenden. Bei Programmdurchlauf erscheint bei Betätigen der Pfeiltasten nämlich kein char-Zeichen, wohl aber eine Integer-Zahl:

```
>> Kap15_19_Gui6_Tecla
c =   und im ASCII-Code: 29
```

Es wurde Pfeiltaste nach rechts (also Cursor nach rechts) gedrückt.

Man kann bzgl. der Pfeiltasten die Funktion dann so aufbauen

```
switch (c)
   case 28
      % Pfeil nach links
      ...
   case 29
      % Pfeil nach rechts
      ...
   case 30
      % Pfeil nach oben
      ...
   case 31
      % Pfeil nach unten
      ...
end
```

Dies soll fürs erste für die GUI-Programmierung reichen.