Einführung in MATLAB GUI

Zum einfachen Erstellen von grafischen Oberflächen bietet MATLAB ein Werkzeug namens 'Graphical User Interface Development Environment', oder kurz guide.

MATLAB-Elemente für GUI-Erstellung:

Was ist ein Handle?
In MATLAB wird jedem grafischen Objekt (wie Button, Eingabefeld, Checkbox usw.) eine eindeutige Zahl („Pointer“) zugewiesen. Diese Zahl wird Handle genannt und dient dazu, auf die grafischen Attribute zugreifen zu können. Viele Funktionen, nutzen Handles, um Objekte eindeutig identifizieren zu können. Die bekanntesten Vertreter sind die Funktionen set (set Object properties) und get (get Object properties) , mit denen Objekteigenschaften verändert bzw. ausgelesen werden können.

Beispiele für Handle:

gcf Get handle to current figure.
gcbo Get handle to current callback object.
gca Get handle to current axis.

Beispiele für get und set:

```
get(gcbo);  % Listet alle Objektattribute
get(gcbo,'String'); % Hole den Inhalt des Attributs 'String'

set (gcbo,'String', 'Hallo Word'); % Setze  'Hallo Word' in 'String'
```

Beispiel Plot:

```
x=0:0.1:2*pi;
plot(sin(x));

h=gca % h ist Handle auf dem Plot

get(h);  % Listet alle Plotattribute
set(h,'Color' ,[0 1 0]); % Aendert das Attribut Color
```

MATLAB-Strukturelemente

Strukturen ermöglichen die Zusammenfassung mehrerer Teilvariablen (in MATLAB als Felder bezeichnet) zu einer Sammelvariablen. Sie können mit Hilfe der Funktion struct oder durch direkte Zuweisung erzeugt werden.

Beispiel

```
AS.name=’Mayer’
AS.geb   =’11.3.1960’
AS.adr    =’Muenchen’
```

Oder

```
AS=struct('name','Mayer', 'geb','11.3.1960','adr', 'Muenchen')
```

AS ist der Strukturname

```
AS;   % Liste den Strukturinhalt
```

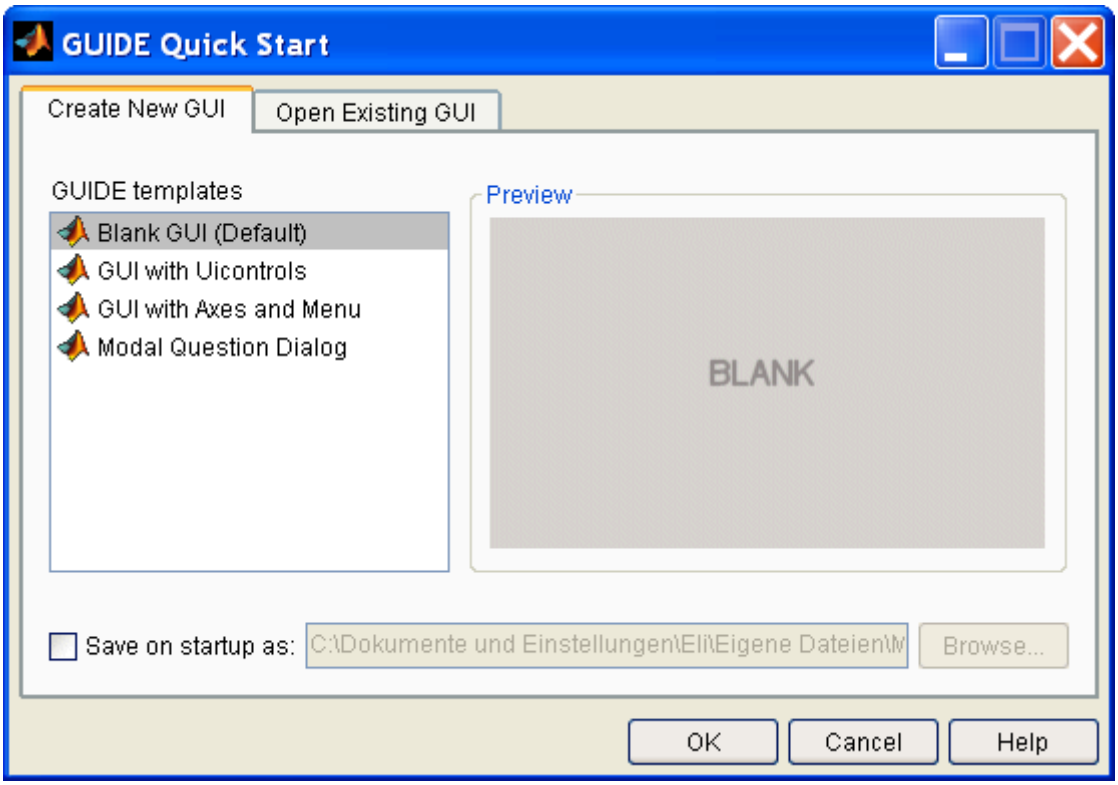
```
> AS =

    name: 'Mayer'
    geb: '11.3.1960'
    adr: 'Muenchen: Dorfstr 16'
```

```
AS.name

AS.geb(1:2)

AS.adr(1:8)


% Umwandlung von String in Zahlen mit str2num MATLAB-Funktion

Geb_tag   = str2num( AS.geb(1:2)      );
Geb_mon =str2num(  AS.geb(4:4)      );
Geb_jahr = str2num(  AS.geb(6:end)  );
```

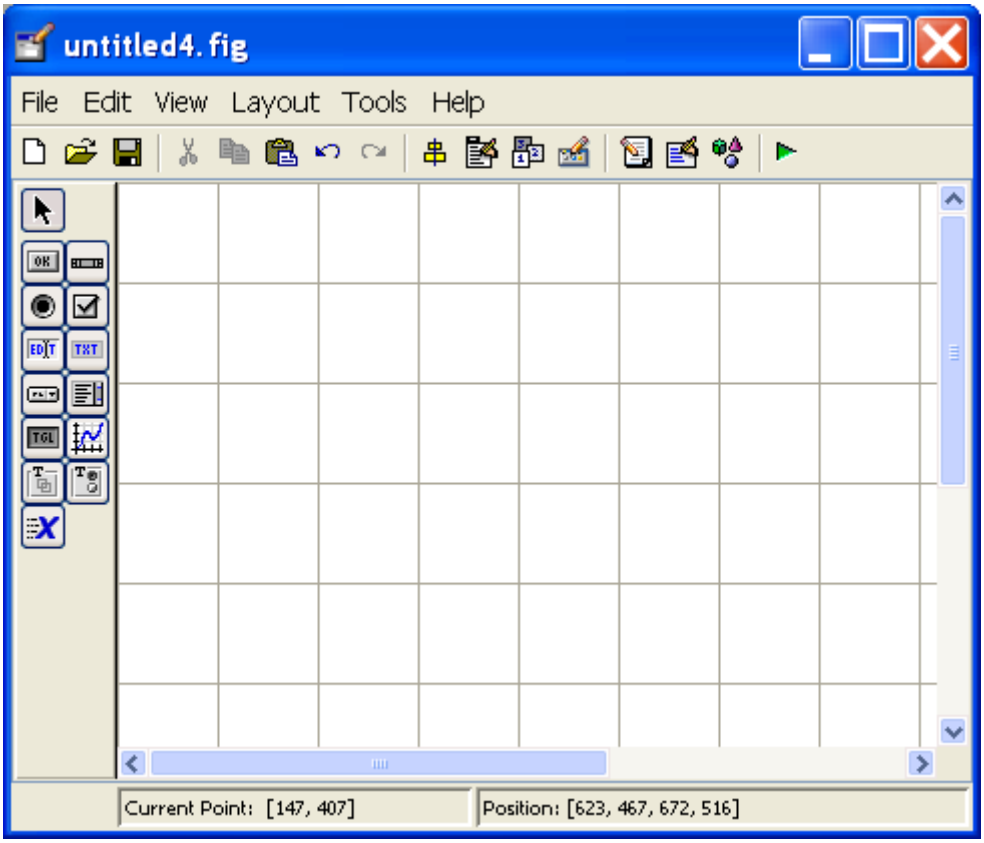
Erstellen einer neuen GUI

Vor dem Erstellen einer GUI, muss guide gestartet werden. Dies kann man über die Kommandozeile mit diesem Befehl tun:

> guide



Es erscheint ein Dialogfenster. Mit 'Blank GUI' kann dabei ein neues, komplett leeres Fenster erstellt werden. Wird diese Auswahl dann mit 'OK' bestätigt, öffnet sich der Dialog-Editor.



Folgende Komponenten sind im Dialog-Editor zu erkennen. Die Toolbar am linken Fensterrand beinhaltet alle Steuerelemente, die für eine GUI verwendet werden können.
Die Toolbar am oberen Fensterrand beinhaltet Funktionen zum Layouten der GUI. Dazu gehört Speichern und Laden der GUI, Kopieren, Einfügen und Ausrichten von Steuerelementen und das Starten der GUI. Alle Aktionen der oberen Toolbar können auch über das Menü ausgeführt werden.

Um Steuerelemente auf die GUI zu setzen, muss zuerst auf das gewünschte Element in der linken Toolbar geklickt werden. Danach kann dieses Element mit einem Klick auf die Fensterfläche in die GUI eingefügt werden. Beispielsweise muss zum Einfügen eines Textfelds zuerst auf den Button mit der Aufschrift 'TXT' geklickt werden und danach auf die Position im Fenster, wo das Textfeld erscheinen soll. Ein neuer Button kann mit einem Klick auf den Button mit der Aufschrift 'OK' hinzugefügt werden. Das Ergebnis könnte dann in etwa so aussehen.

Jedes Element hat individuelle Eigenschaften. Um diese zu verändern gibt es den Property Inspector. Der Property Inspector kann über einen Doppelklick auf das zu verändernde Element geöffnet werden.

Hier sieht man die Eigenschaften des Buttons. Die Eigenschaft 'String' gibt dabei den Text an, den das Steuerelement anzeigen soll. 'Tag' ist der Name des Steuerelements und 'Position' ist die Position und die Größe des Elements auf der GUI.
Wird beispielsweise die Eigenschaft 'String' des Buttons auf 'Start' geändert, so ändert sich analog dazu auch der angezeigte Text in der GUI.

Hinzufügen von Funktionalität

Der Code, an dem der Cursor sich befindet, sieht in etwa so aus.

```
% --- 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)
```

Beim Klick auf den Button wird somit die Funktion pushbutton1_Callback aufgerufen. Beim Aufruf werden folgende Daten an die Funktion übergeben: "hObject" repräsentiert ein Handle auf das Button-Objekt selbst. "eventdata" wird zur Zeit nicht von MATLAB verwendet und ist somit leer. "handles" ist eine Struktur, in der sich die Handles auf jedes Steuerelement auf der GUI befinden. Auf das jeweilige Handle kann mit dem Objektnamen (also dem Wert im Feld 'Tag' des Property Inspectors) zugegriffen werden. So liefert beispielsweise "handles" das Handle auf den Button.
Um die Eigenschaften eines GUI-Elements zu verändern, gibt es den Befehl set. set erwartet als Eingabeparameter das Handle des Objekts, das geändert werden soll, danach die Eigenschaft, die geändert werden soll und den Wert, den die Eigenschaft annehmen soll. Die verfügbaren Eigenschaften eines Elements können zum größten Teil im Property Inspector nachgesehen werden. Um etwa den Text des statischen Textes zu verändern, schreiben wir die Callback-Funktion folgendermaßen um:

```
% --- 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)
set(handles.text1, 'String', 'Hallo Welt');
```

Starten der GUI Nach dem Abspeichern und Schließen der Datei kann die erstellte grafische Oberfläche gestartet werden. Entweder über den grünen Pfeil in der oberen Taskleiste, mit Strg+T oder über den Menüeintrag "Tools → Run".

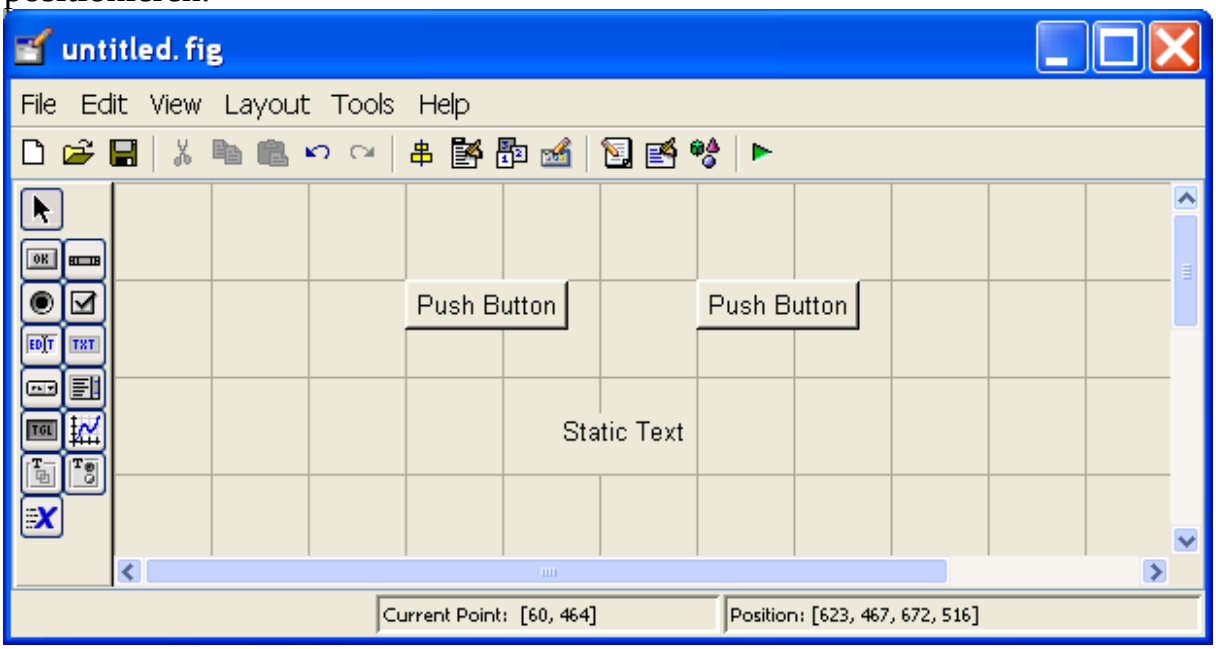
Und dann erscheint die GUI in ihrer vollen Pracht. Zugegeben, sie ist noch etwas spartanisch, aber es sollte jetzt möglich sein, weitere Steuerelemente darauf zu platzieren und auf Knopfdruck zu verändern.

Beispiel 1:

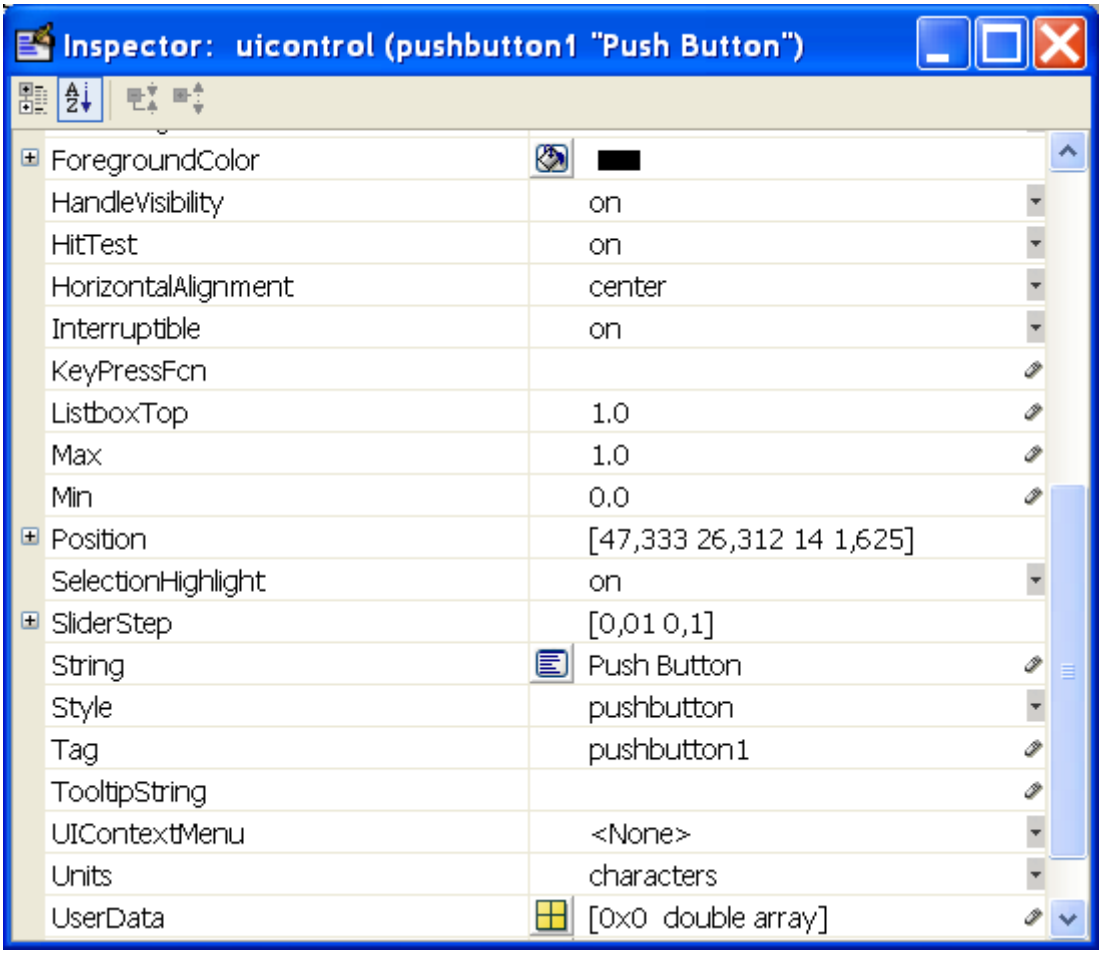
Es soll ein GUI mit zwei Knöpfen „ Plus“ und „Minus“ erstellt werden.
Bei Betätigung des „Plus“ Knopfes soll ein Zähler um Eins erhöht werden, bei der Betätigung des „Minus“ Knopfes soll der Zähler um Eins erniedrigt werden. Der aktuelle Zählerstand soll auf der GUI visualisiert werden.

Vorgehensweise:
> guide
Blank GUI (Default) auswählen

1) Zwei „Push buttons“ und ein „Static Text“ Element holen und am GUI positionieren.



2) Bei Doubleklick auf den linken Push Button erscheint eine Maske mit den Objektattributen (Inspector).



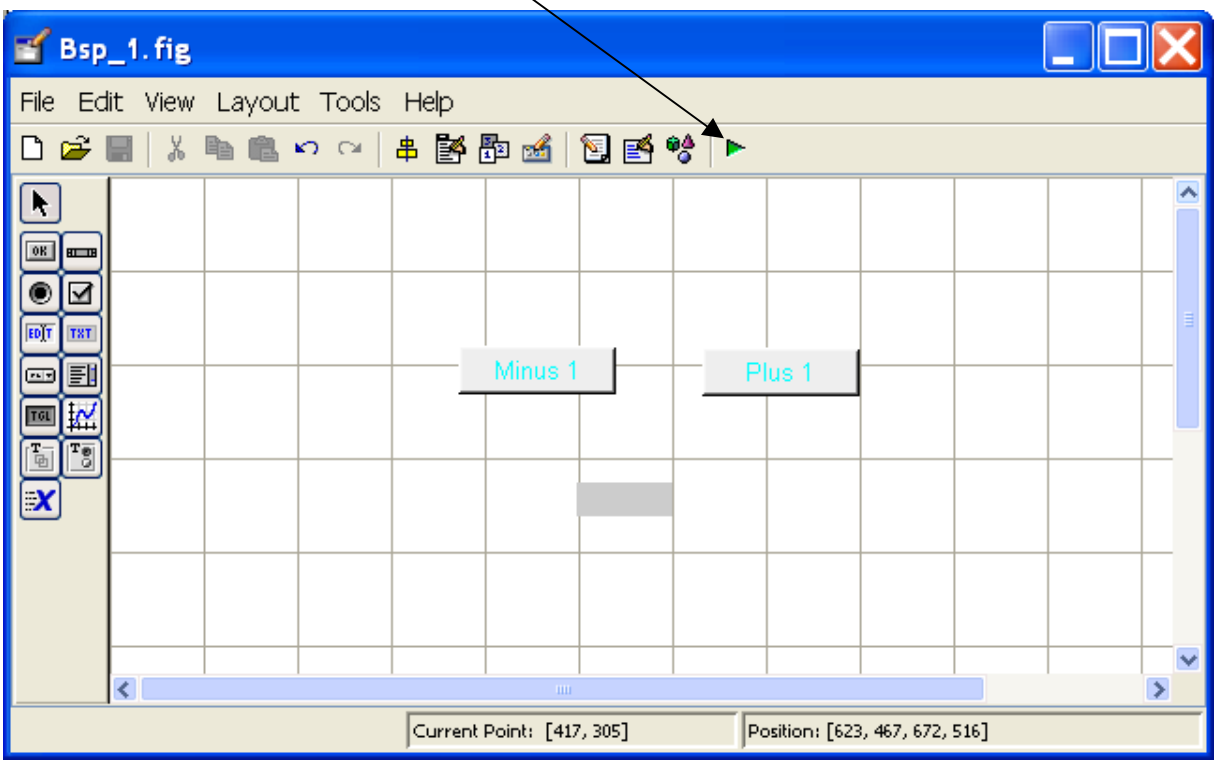
Das Attribut „String“ Default text „Push Button“ wird zu „Plus1“ geändert.
Das Attribut „Tag“ Default text „Pushbutton1“ wird zu „Plusknopf“ geändert.

Analog werden die Attribute von dem rechten Knopf zu:
String = „Minus1“ und Tag = “Minusknopf“.

Das „Tag“ Attribut vom „static Text“ wird zu „Ausgabe“ geändert.

3) Die GUI File wird nun gespeichert : Bsp_1.fig (save as → bsp_1)

4) Testen mit dem „Grünen Pfeil „



5) Implementierung der Anwendung:

Die GUI hat folgende Strukturelemente

handles.Minusknopf
handles.Plusknopf
handles.Ausgabe

MATLAB generiert beim Speichern der GUI eine m-file (Hier Bsp_1.m). In dieser File wird die Anwendung implementiert.

Erläuterungen:

Die Lila Zeilen sind die Implementierung der Anwendung, Alle andere Zeilen wurden generiert.

Im Teil Executes just before Bsp_1 is made visible
wird die Struktur handles um handles.z (Zähler Variable) erweitert und auf Null initialisiert.

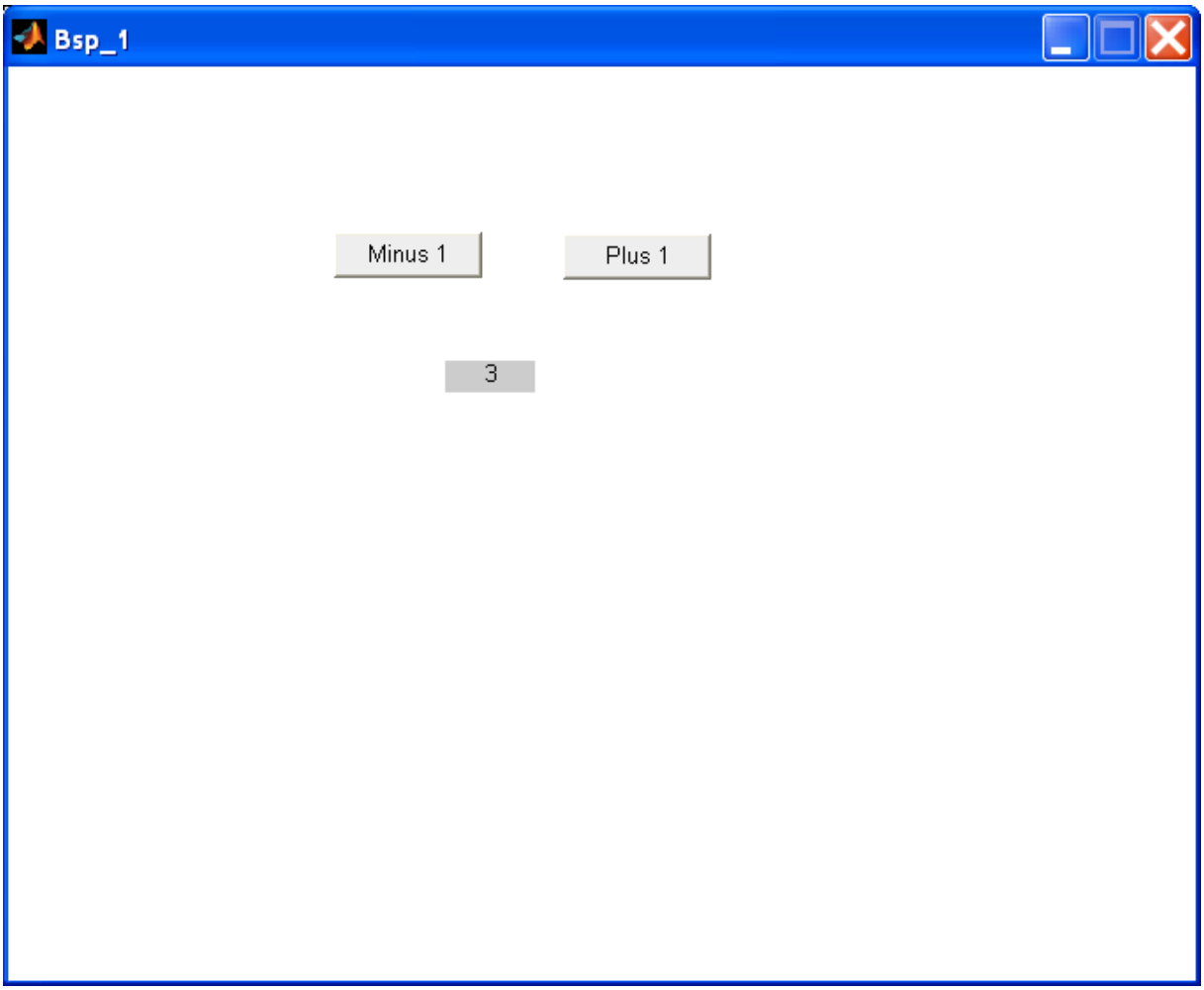
Im Teil Executes on button press in Plusknopf
wird der Zähler um Eins erhöht, dann werden die Daten gespeichert und mit dem Set Befehl ausgegeben.
Analog wird der Executes on button press in Minusknopf ergänzt

Generierte Bsp_1.m file:

```
% --- Executes just before Bsp_1 is made visible.
function Bsp_1_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 Bsp_1 (see VARARGIN)


% Choose default command line output for Bsp_1
handles.output = hObject;


% Zaehler Initialisierung
handles.z = hObject;
handles.z=0;

% Update handles structure
guidata(hObject, handles);


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

handles.z=handles.z+1; % Zaehler +1
guidata(hObject, handles % Update handles structure
set(handles.Ausgabe,'String',num2str(handles.z));% Ausgabe
```

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

handles.z=handles.z-1;       % Zaehler -1
guidata(hObject, handles);   % Update handles structure
set(handles.Ausgabe,'String',num2str(handles.z)); % Ausgabe
```

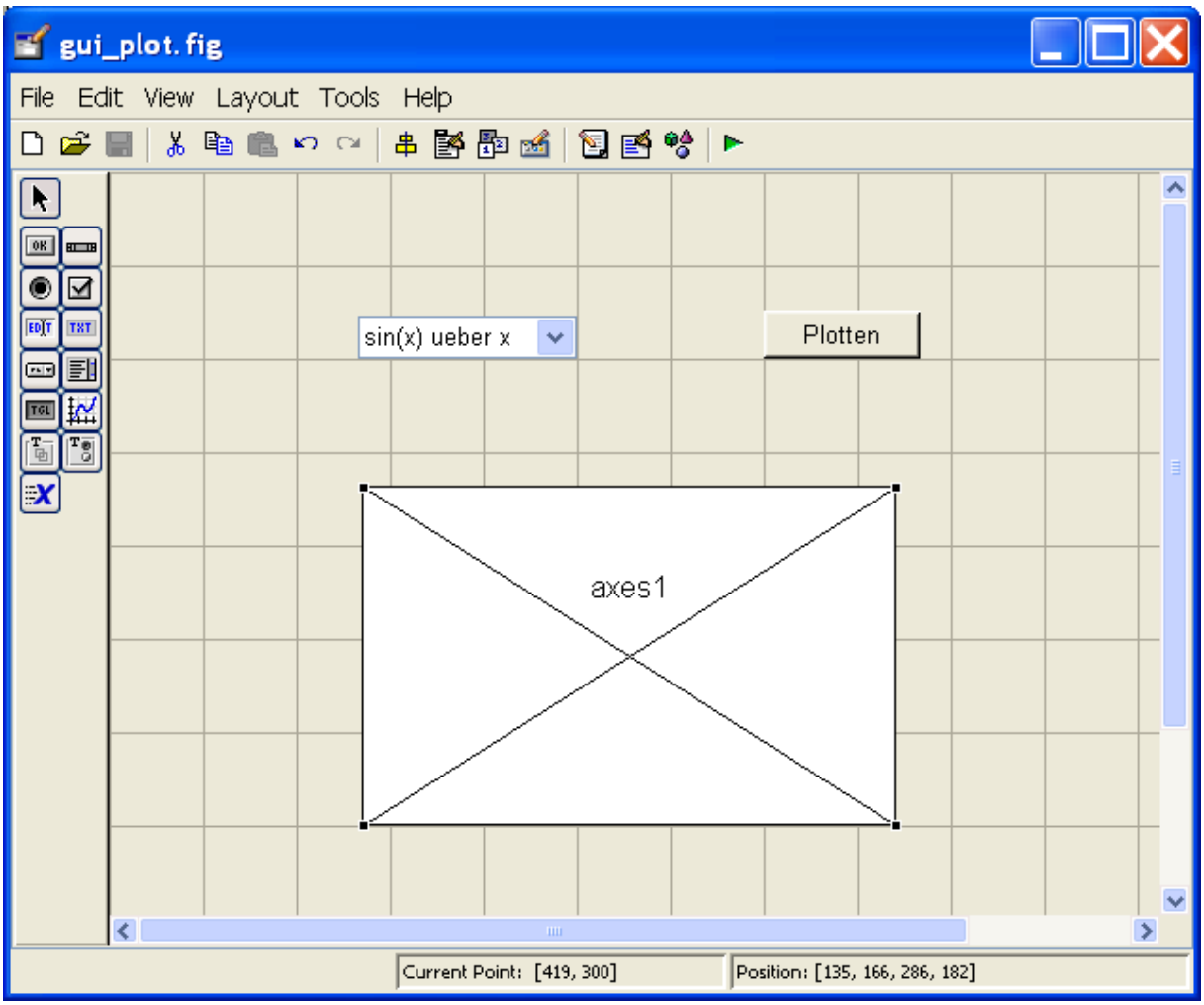
Ergebnis:
Starten in MATLAB Command Window
> Bsp_1(1)



Beispiel 2:

Der Anwender soll aus einem Auswahlfenster (Popup Menu) eine Funktion auswählen und plotten.

gui_plot.fig



MATLAB gui_plot.m

Die Lila Zeilen sind die Implementierung der Anwendung, Alle andere Zeilen wurden generiert.

```
% --- Executes just before gui_plot is made visible.
function gui_plot_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 gui_plot (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);

% Plot Initialisierung
if strcmp(get(hObject,'Visible'),'off')
    x=0:0.1:2*pi;
    plot(x,sin(x));
    grid
    title('sin(x) ueber x')
end

% --- 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)

axes(handles.axes1); % Plot Fenster erzeugen
cla;                 % Plot Inhalt Loeschen

St=get(handles.Plotauswahl,'String');% Plot Auswahl holen (String fuer title)

popup_sel_index = get(handles.Plotauswahl, 'Value'); % Welche Plot wuerde
Ausgewaehlt


switch popup_sel_index
    case 1
        x=-2*pi:0.1:2*pi;
        plot(x,sin(x));
        grid
        title(St(1));
```

```
    case 2
        x=-2*pi:0.1:2*pi;
        plot(x,cos(x));
        grid
        title(St(2));
    case 3
        x=-pi/4:0.1:pi/4;
        plot(x,tan(x));
        title(St(3));
        grid
end



% --- Executes on selection change in Plotauswahl.
function Plotauswahl_Callback(hObject, eventdata, handles)
% hObject    handle to Plotauswahl (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 Plotauswahl contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from Plotauswahl
;


% --- Executes during object creation, after setting all properties.
function Plotauswahl_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Plotauswahl (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

% Plot Auswahl
set(hObject, 'String', {'sin(x)', 'cos(x) ', 'tan(x) '});
```

Aufgabe:

Teil 1:

Erstellen Sie ein GUI mit einem Knopf mit der Beschriftung „Plotten“ und einem Plotfenster. Bei der Betätigung des Knopfes soll die Funktion

$$x(t) = R * \cos(\omega * t)$$
$$y(t) = R * \sin(\omega * t)$$
$$z(t) = \alpha * t$$

mit:

$$\omega = \frac{2\pi}{T}$$
$$\alpha = \frac{\omega * h}{2\pi}$$

geplottet werden.

Daten:

h=4, T=86400,R=1;
t=-10:0.1:10;

Hinweis:

- Starten Sie die GUI Entwicklungstools guide
- Holen Sie ein „Pushbutton“ und ein „Graph Element“
- Vergrößern und positionieren Sie die Elemente
- Speichern Sie die Figur unter der Name Helix.fig
- Beim Speichern sehen Sie im MATLAB Editor der File Helix.m
- In dieser File wird die Anwendung implementiert.

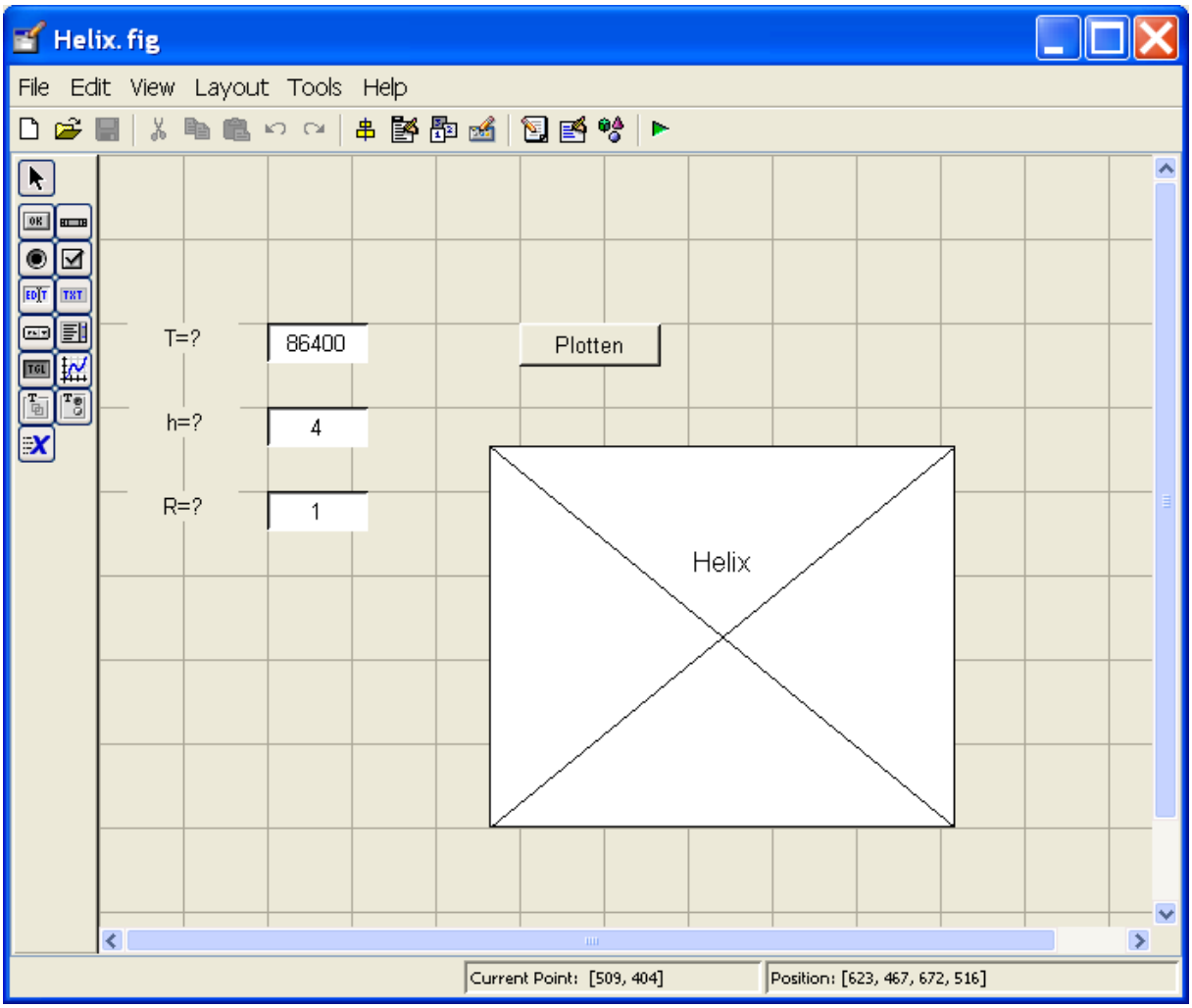
Um die Funktion zu plotten benutzen Sie die MATLAB-Funktion plot3(x,y,z).

Teil2:

Ergänzen Sie den GUI so, dass die Funktionsparameter (T, h, R) mit Hilfe von „Edit Text“ Element eingegeben werden können.

Hinweis:

- Erzeugen Sie diese GUI



- Die Defaultdaten (T=86400, h=4, R=1) können als „String“ des Elements angegeben werden.

- Vergeben Sie für den Parameter T „Tag-Name“ T, für den Parameter h Tag-Name“ h, und für R „Tag-Name“ R.

Damit wird es leichter, in dem generierten Programm die Handles zu identifizieren. Zum Beispiel: der geänderte T Parameter wird mit dem Befehl T=get(handles.T,'String'), geholt und steht als String zur Verfügung. Mit dem Befehl T=str2double(T), wird der String T in eine Zahl umgewandelt.

- In dem Programmteil : % --- Executes just before Helix is made visible, wird die Initialisierung des Plots implementiert

- In dem Programmteil : % --- Executes on button press in Plotten, wird der Plot mit den aktuellen Parametern implementiert.

Ergebnisse: