Chapter 6

Welcome to Carabao Island

*San Jose is a fifth-class* [*municipality*](https://en.wikipedia.org/wiki/Philippine_municipality) *in the* [*province*](https://en.wikipedia.org/wiki/Philippine_province) *of* [*Romblon*](https://en.wikipedia.org/wiki/Romblon)*,* [*Philippines*](https://en.wikipedia.org/wiki/Philippines)*. Its territory is contiguous with Carabao Island, otherwise known as Hambil Island, the southernmost island of the province. According to the 2010 census, it has a population of 10,294 people.*

[](http://www.google.at/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiz3uXG17HKAhWGfg8KHQMpC0EQjRwIBw&url=http%3A%2F%2Fmapio.net%2Fa%2F111965201%2F&bvm=bv.112064104,d.ZWU&psig=AFQjCNEiwqHHTdyJO68eABLHl8yumVvznQ&ust=1453148863990294)The *Carabao* shell provides a menu driven interactive graphical user interface. A main goal of the *Carabao* rapid prototyping approach is to build quickly powerful shells based on available building blocks. It is worth to have a separate chapter which is devoted solely to shell construction, and the reader who seriously wants to improve his skills of rapid shell prototyping is strongly advised to read this chapter with utmost attention. We have already a good understanding of *Carabao* shell functionality from a user's view, and we got some impression how a rapid prototyped shell looks like, but frankly speaking, we still do not have a solid understanding of the powerful possi­bilities of shell prototyping by utilizing *Carabao* methods and *Carabao* building blocks. This is the topic of this chapter.

# Charm

There is a third menu functionality, called a 'charm menu item'[[1]](#footnote-1) which allows to edit any character string or scalar double value setting.

>> setting(o,{'play.value'},pi); % default setting for value

>> ooo = mitem(oo,'Value',{},'play.value'); % add menu item

>> charm(ooo,{}); % add charm functionality

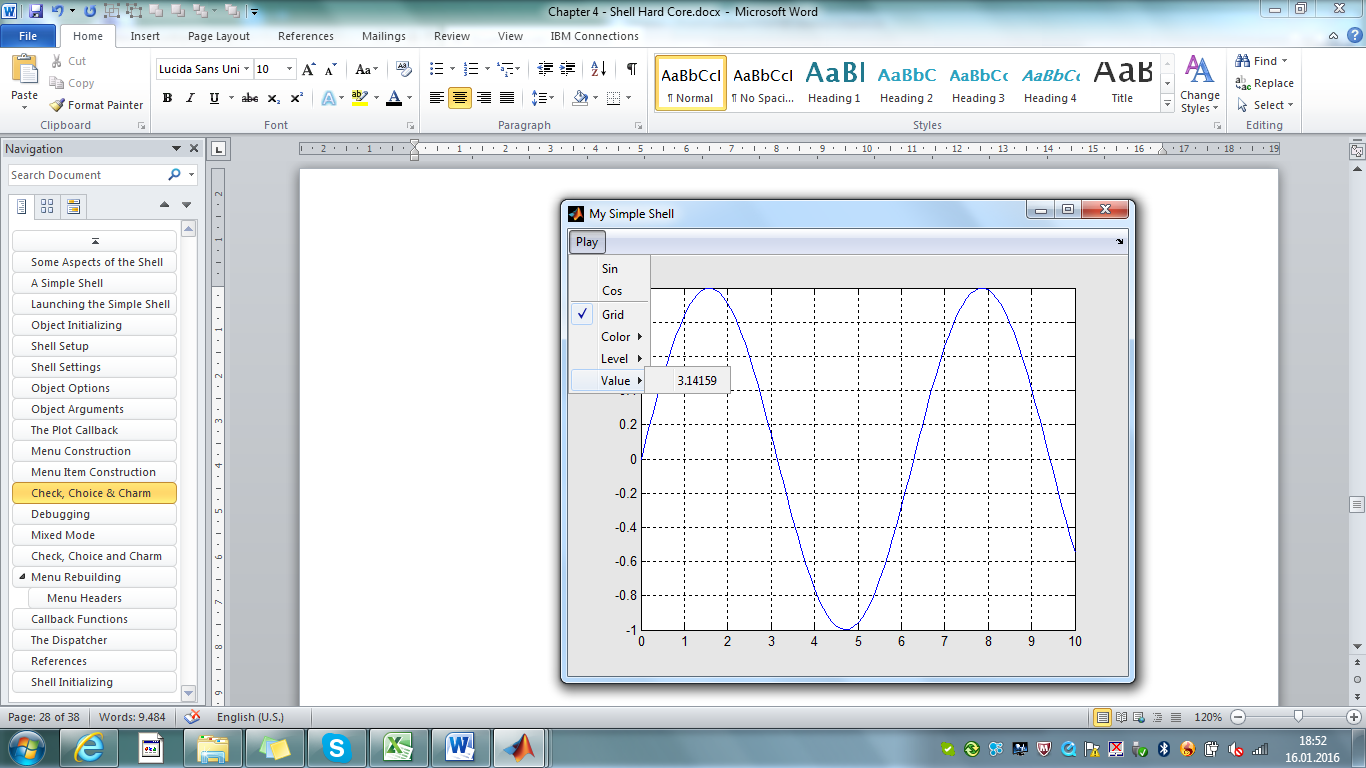


Fig. 4.7 – Tiny menu with *check*, *choice* and charm menu items

Upon clicking on *Play>Value* an edit dialog opens which allows to modify the value of the shell setting which is controlled by the charm menu item.

Let us investigate the object returned by *menu>Begin*. Obviously new fields, *figure* and *mitem,*  have been added to the *work* property, and we see additional information 'MITEM Details'.

>> o=menu(o,'Begin') % popup an empty shell

ESPRESSO object

MASTER Properties:

: :

WORK Property:

opt: [1x1 struct]

figure: 1

mitem: 1

MITEM Details

menu item: #1 (figure)

>> o % investigate object internals

ESPRESSO object

MASTER Properties:

tag: espresso

type: shell

par:

title: 'My Simple Shell'

data: {}

WORK Property:

opt: [1x1 struct]

Note that the *refresh* callback setup has added an *opt* field (options) to the *work* property of the object. Now pop up an empty shell figure.

>> o=menu(o,'Begin'); % popup an empty shell figure

This command calls local function *menu>Begin* [[2]](#footnote-2). A new figure titled 'My Simple Shell' with an empty menu bar pops up. Let us investigate the object returned by *menu>Begin*. Obviously new fields, *figure* and *mitem,*  have been added to the *work* property, and we see additional information 'MITEM Details'.

>> o=menu(o,'Begin') % popup an empty shell

ESPRESSO object

MASTER Properties:

: :

WORK Property:

opt: [1x1 struct]

figure: 1

mitem: 1

MITEM Details

menu item: #1 (figure)

Work property field *figure* contains always the figure handle of the related shell, and there are two ways to retrieve it from the object.

fig=o.work.figure; % get figure handle (fast)

fig=figure(o); % get figure handle (recommended)

The other important graphics handle carried by the object is the actual menu item handle, and there are also two ways to set it and retrieve it from the object.

o.work.mitem=hdl; % set object's menu item handle (fast)

o=mitem(o,hdl); % set object's menu item handle (recommended)

hdl=o.work.mitem; % get object's menu item handle (fast)

hdl=mitem(o,inf); % get object's menu item handle (recommended)

Now enter the following command line.

>> o=mitem(o); % sets mitem handle equal to figure handle

This command is a bit redundant in this context, as it has been executed already at the end of *menu>Begin.* Anyway it is important to understand the principle: the command copies the object's figure handle into the object's menu item handle. If we enter now

>> oo=mitem(o,'Play'); % add Play menu item at top level

then the following sequence of actions will happen.

hdl=uimenu(mitem(o,inf),'Label','Plot'); % add menu item

oo=mitem(o,hdl); % store new menu item handle in object oo

With mitem(o,inf)the actual menu item handle is retrieved from object *o* and used as the first argument of the *uimenu* call (the handle of the parent graphics object), which creates the menu item. The handle of the new menu item *hdl* is then stored back to object *oo* which is based on a copy of object *o*. This game can be continued for the whole menu hierarchy.

>> ooo=mitem(oo,'Sin',{@Plot,'sin'}); % add Sin menu item

>> ooo=mitem(oo,'Cos',{@Plot,'cos'}); % add Cos menu item

These lines translate into

callback=call(o,o.tag,{@SinCb}); % compose callback

hdl=uimenu(mitem(o,inf),'Label','Sin','Callback',callback);

oo=mitem(o,hdl); % store new menu item handle in oo

callback=call(o,o.tag,{@CosCb}); % compose callback

hdl=uimenu(mitem(o,inf),'Label','Cos','Callback',callback);

oo=mitem(o,hdl); % store new menu item handle in oo

The object variables *o*, *oo*, *ooo*, … indicate in a natural way the menu hierarchy level. Complete the menu construction with

>> menu(o,'end'); % end menu setup

The underlying local function *menu>End* (precisely: *menu@carabao>End* ) will finally refresh the screen and display some object information of fig. 37 ('My Simple Shell').

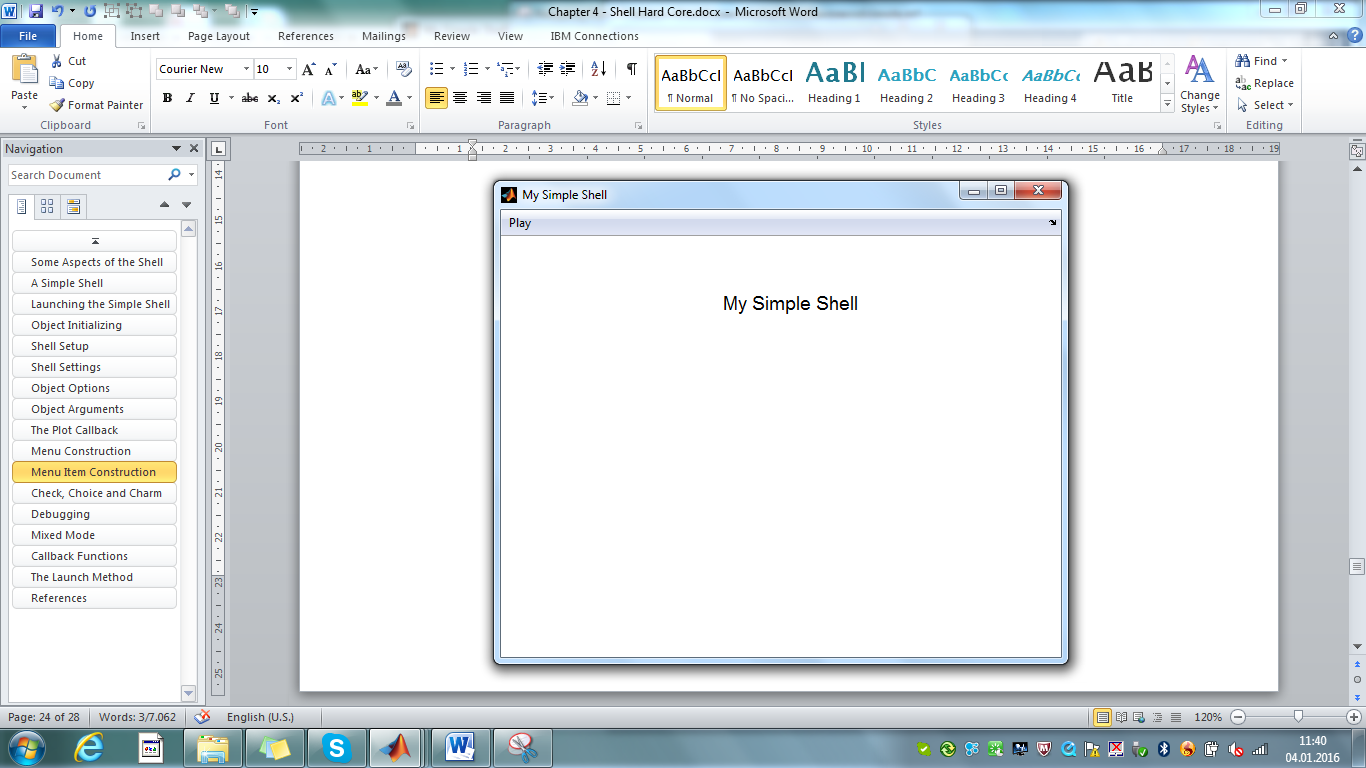


Fig.37 – the initial screen

Note that the callbacks are not working since we have constructed the @Plot function handle from the command line which makes it impossible to access a local function[[3]](#footnote-3).

# Debugging

There is a nice possibility to display some detail information of the actual menu item. This feature is helpful in debug phases. Invoke the following commands.

>> ooo=mitem(oo,'Tan',{@TanCb},'b','enable','off');

>> mitem(ooo)

menu item: #1 / Plot / Tan

callback: {@call [1x1 espresso] {@TanCb}}

userdata: b

visible: on

enable: off

The information displayed by >> mitem(ooo) tells us the path of the menu item (*figure #1>Plot/Tan* ) and displays the values of the most interesting properties. These 'MITEM Details' are automatically displayed as object internals, when a menu item handle is provided in the work properties.

# Mixed Mode

Finally it is worth to mention that there is no reason why mitem and uimenu calls cannot be mixed. This makes sense if code for menu construction exists already and should be integrated into the *Carabao* frame work. Study the following example.

function oo = Shell(o) % Shell setup

o = Init(o); % initialize object

o = menu(o,'Begin'); % begin menu setup

oo = menu(o,'File'); % add File menu

oo = Plot(o); % add Plot menu

ooo = mitem(oo,'Tan',{@TanCb}); % add Tan menu item to Plot menu

o = menu(o,'End'); % end menu setup

end

function oo = Plot(o) % Plot Menu (existing code)

men = mitem(o,inf); % get menu item handle

sub = uimenu(men,'label','Plot'); % add Plot menu header

uimenu(sub,'Label','Sin','Callback'{@SinCb});

uimenu(sub,'Label','Cos','Callback'{@CosCb});

oo = mitem(o,sub); % object with Plot menu handle

function SinCb(object,event)

: :

end

function CosCb(object,event)

: :

end

end

# Check, Choice and Charm

# Menu Rebuilding

## Menu Headers

# Callback Functions

# The Dispatcher

* ca

# References

[1] *Stormy Attaway*: "MATLAB® (2013) – A Practical Introduction to Programming and Problem Solving" (3rd edition); Butterworth-Heinemann, Elsevier Inc. 2013, ISBN: 978-0-12-405876-7

[2] *MATLAB*® – Object-Oriented Programming – R2015b; Mathworks, online on the internet

The following conditions must hold in order that the rebuild mechanism for the menu can work properly.

* The object which is provided as the first argument must have refreshed options matching the actual shell settings. This is achieved by pulling the object from the shell.
* The underlying local function (simple>*Play* ) must allow multiple calls, otherwise multiple instances of menu *Play* would be constructed. It also means that initialization of shell settings done by such function must be conditionally (only at the first time)
* Local function *simple>Play* must be a so called *managed function*, i.e, a local function which can be invoked from outside the method (*simple*).

We will learn more about object options in the next section. The second aspect that the underlying local function must allow multiple calls, implies that the local function does not change shell settings which are already initialized. On the other hand it is good *Carabao* programming style to initialize shell settings related to a sub-menu in the function which constructs the sub-menu, as it is the case with *simple>Play*. Thus there is a common need to initialize a shell setting conditionally (only if it is currently not initialized, i.e. the current value is empty). The following calling syntax of *setting* does exactly this job.

setting(o,{tag},value); % init setting if not initialized

# Shell Initializing

Before we finish our introduction to object options we have to highlight the importance of options during the initializing process of the shell. Close all figures (*File>Exit*), launch a Simple shell (>> simple(espresso);), plot a cosine curve (Play>Cos), select green color (*Play>Color>Green*) and grid (*Play>Grid*). A green cosine curve with a grid will finally displayed on the shell's screen (fig. 4.3). Now study the shell settings.

>> setting(espresso) % display shell settings

control: [1x1 struct]

play: [1x1 struct]

>> play=setting(espresso,'play') % retrieve shell settings 'play'

play =

grid: 1

color: 'g'

Everything happens due to expectation. Now pull the shell object from the shell into MATLAB work space (>> o=pull(espresso);) and investigate the options.

>> o=pull(espresso); % pull object from shell

>> opt(o) % display options

control: [1x1 struct]

play: [1x1 struct]

>> play=opt(o,'play') % retrieve option 'play'

play =

grid: 1

color: 'g'

The options are matching the shell settings (as expected) [[4]](#footnote-4). Now close the shell (*File>Close*). What if we launch again a shell for the object? Go and try it!

>> launch(o); % launch a shell for object o

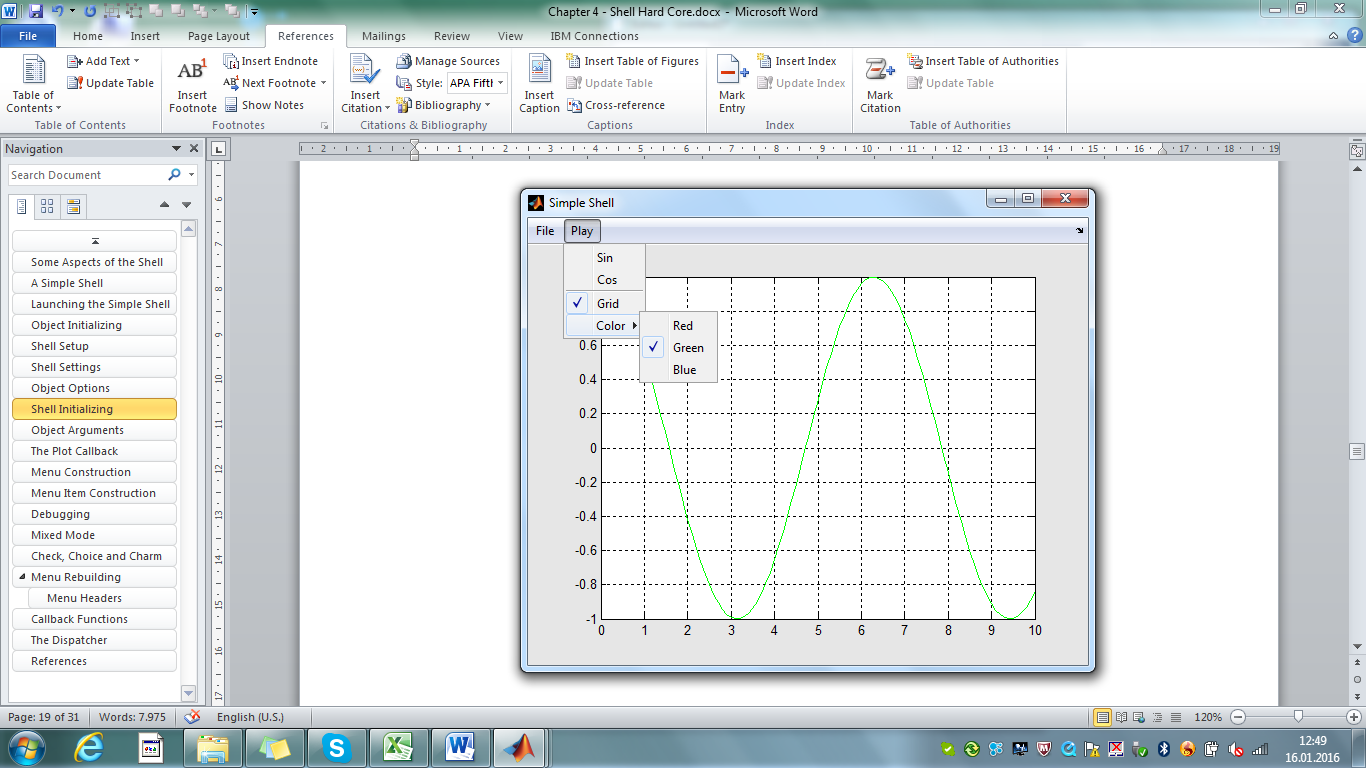
The shell re-appears in the same outlook as it looked when we closed the shell[[5]](#footnote-5): a green cosine curve with grid! Even the check marks in the Play menu are recovered consistently (fig. 4.4). Are you surprised now?

Fig. 4.4 – The *Simple* shell recovers consistently

Probably not, if you started internalizing the *Carabao* philosophy. Once the shell object is pulled from the shell it contains all information to reconstruct both the internal data structures of the shell as well as the outlook of the shell (that has been at the time when the object was pulled from the shell).

The mechanism behind is straight forward. When the object is pulled from the shell all shell settings are copied into the object options. Thus the object holds already all required information for shell reconstruction. The only piece that is missing is how the shell method initializes the shell settings during the initialization process of the shell.

Study the setup procedure of the *Simple* shell! After object initialization (*Init* ) the local function *carabao/menu>Begin* is consulted.

function o = Shell(o) % Shell Setup

o = Init(o); % initialize object

o = menu(o,'Begin'); % begin menu setup

oo = File(o); % add File menu

oo = Play(o); % add Play menu

o = menu(o,'End'); % end menu setup

end

And at a deeper level of *carabao/menu>Begin [[6]](#footnote-6)*  the following code lines could be found where after opening a new figure the full set ('bag') of options are retrieved from the object and used to initialize the shell settings[[7]](#footnote-7).

opts = opt(o); % full bag of options

setting(o,opts); % initialize settings with options

o = push(o); % push object into figure

After that the object is pushed into the shell which completes recovery of the previous shell state. The following steps of the shell setup (adding the File and Play menu) will be based on both the actual shell settings and the shell object's properties. Finally local function *carabao/menu>End [[8]](#footnote-8)* invokes the refresh method to get the prepared refresh function executed, which in our case plots the cosine curve.

This procedure emphasizes once more that the initializing function *Init,* which is executed before the actual menu setup, must not change object properties which are already initialized (non-empty). If, e.g., we want our *Simple* shell to come up with a blue sine curve after launching an uninitialized *Espresso* object, we have to modify the *Init* function as follows.

function o = Init(o) % Object Initializing

o = dynamic(o,false); % setup as a static shell

o = launch(o,mfilename); % setup launch function

o = provide(o,'par.title','Simple Shell');

o = provide(o,'par.comment',{'A simple shell to play'});

o = refresh(o,{@Plot,'cos'}); % provide refresh callback

o = opt(o,{'play.color'},'b'); % provide blue plot color

end

Note that the syntax for providing the *refresh* callback causes to change the control option control.refresh only if it is not initialized, and if that is the case the callback {@Plot,'cos'} is provided which plots a cosine curve. The color of the cosine curve is prepared by the *play.color* option. The special syntax o = opt(o,{'play.color'},'b') changes the option setting only if the option is not initialized. The alternative

o = opt(o,'play.color','b'); % force blue color setting

would force to have always blue color setting after shell launch.

1. why has the *charm* menu item such a funny name? the answer might be an eternal puzzle like in particle physics where it is also hard to say why the *charm quark* got its funny name [↑](#footnote-ref-1)
2. actually: *menu@carabao>Begin* [↑](#footnote-ref-2)
3. there would be, however a possibility to make the callbacks working.

   >> ooo=mitem(oo,'Sin',{'simple','Plot','sin'}); % add Sin menu item

   >> setting(o,{'play.color'},'r'); % initialize setting [↑](#footnote-ref-3)
4. we did not compare the control settings with the control options, but you can check and you will prove that they are identical [↑](#footnote-ref-4)
5. More precisely: when we pulled the object from the shell [↑](#footnote-ref-5)
6. *carabao/menu>Begin* is invoked by menu(o,'Begin') [↑](#footnote-ref-6)
7. if you really want to dig down to this code you have to investigate *caracow/menu>Begin*. *caracow* is one of the super classes of *carabao* [↑](#footnote-ref-7)
8. *carabao/menu>End* is invoked by menu(o,'End') [↑](#footnote-ref-8)