WWD

Web Window Driver

Martin Saurer, 29.05.2013

Table of Contents

Foreword

§1. First, I have to say that English is not my native language. So, I apologize for any strange looking phrases and grammatical gaffes. If it's anybody out there who is able to translate this manual into correct and nice looking English, please feel free to do so.

Volunteers are always welcome.

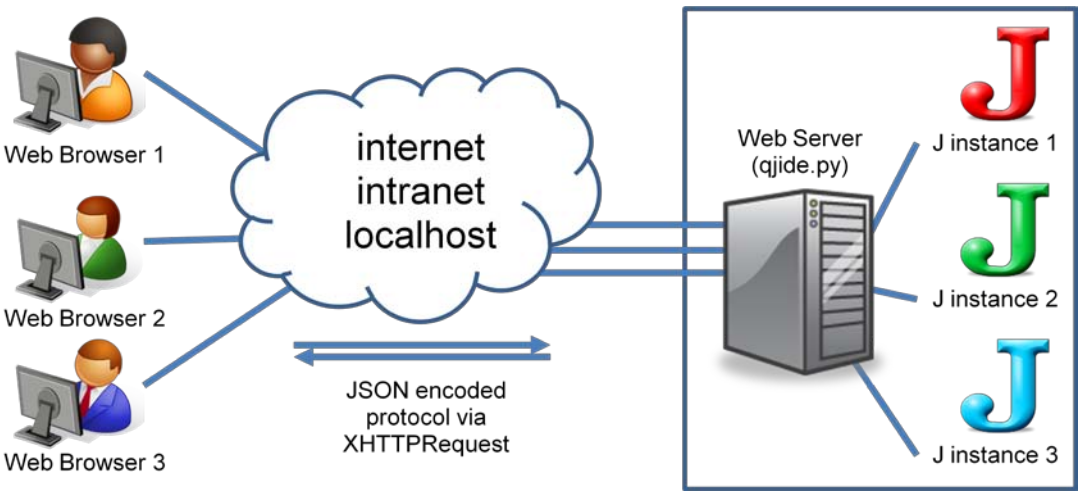
§2. When writing a Web Window Driver (WWD) with a browser frontend and web server backend, there are some restrictions by nature. Please don't compare J's Window Driver (GTK or Qt) to WWD. These are not the same, and never will be. Nevertheless, WWD allows one to write a web frontend (that lives in the web browser) in pure J. No HTML, CSS, JavaScript, etc. is necessary.

§3. WWD is based on Qooxdoo (www.qooxdoo.org), the same AJAX framework as for qjide, the "Qooxdoo J Integrated Development Environment". WWD abstracts the complexity of Qooxdoo, but offers only a subset of its features. More features and widgets may be implemented at a later stage.

§4. Caution: I don't know whether or not, WWD is ready for production use. All my tests on all platforms were successful, but keep in mind that WWD has not currently passed a real field test with real users. So please be careful when use WWD in a production environment. According to the GPL3, I'm not responsible for any damage, loss of goods, money, lives, etc. when using "qjide" or "WWD".

The Basics

A web browser always talks to a web server using the network, regardless whether both (server / client) run on the same machine or not. By nature, browsers implement some security features which are required to handle properly. So for example, a web browser is not allowed (by default) to access the local file system. The web server (of course) must be able to do this. Take a look at the schema below, to understand how qjide and WWD works.



To clarify:

- All J instances, that mean j.dll, libj.dylib or libj.so, run inside the same process, the Python based web server (qjide.py).

- If JMulSes is set to True (JMulSes = True) in qjide.cfg, each J instance runs in its own thread.
  If JMulSes is set to False (JMulSes = False) in qjide.cfg, each client uses the same J instance.

- Separate J threads are started for different client IP addresses.

- The frontend (web browser) portion of WWD is responsible for displaying the widgets, and firing events to the backend (web server).

- The J portion of the WWD, encodes J sentences (JSON) so they can be sent to the frontend.

WWD Verbs

The J part of qjide (qjide.ijs) defines five verbs for WWD:

```
wwd
wwdbeg
wwdadd
wwdend
wwdget
```

(See chapter: “Event Handlers”)

WWD verbs and their arguments:

All WWD verbs take at least 2 arguments (parameters), the command-string, and the command parameters (arguments).

- A single WWD command has the form:

```
wwd '<command-string>' ;< arg_1;arg_2;…;arg_n
```

- Multiple WWD commands must be given in the form:

```
wwdbeg ''
wwdadd 'command-string' ;< arg_1;arg_2;…;arg_n
wwdadd 'command-string' ;< arg_1;arg_2;…;arg_n
…
wwdend’’
```

- Never use multiple WWD commands like this:

```
wwd 'command-string' ;< arg_1;arg_2;…;arg_n
wwd 'command-string' ;< arg_1;arg_2;…;arg_n
```

WWD argument types:

There are three types of arguments, depending on widget: Single Value, Record, and Table.

Single value examples:

```
10

3.141

'Hello World'
```

Record examples:

Use always boxed arrays for record parameters.

```
+-+-+-+-+-+-+-+-+-+-+
|0|1|2|3|4|5|6|7|8|9|
+-+-+-+-+-+-+-+-+-+-+

+-----+-----+------+
|Hello|World|to all|
+-----+-----+------+
```

Table examples (caution!):

Table parameters must always be passed in the following form:

```
+-------+-------+-------+
|+-+-+-+|+-+-+-+|+-+-+-+|
||0|1|2|||3|4|5|||6|7|8||
|+-+-+-+|+-+-+-+|+-+-+-+|
+-------+-------+-------+
```

Assuming there is a table to pass to WWD like this one:

```
<"0 i. 3 3

+-+-+-+
|0|1|2|
+-+-+-+
|3|4|5|
+-+-+-+
|6|7|8|
+-+-+-+
```

Simply re-shape it, using:

```
<"1 <"0 i. 3 3
```

to get the required shape:

```
+-------+-------+-------+
|+-+-+-+|+-+-+-+|+-+-+-+|
||0|1|2|||3|4|5|||6|7|8||
|+-+-+-+|+-+-+-+|+-+-+-+|
+-------+-------+-------+
```

Widgets

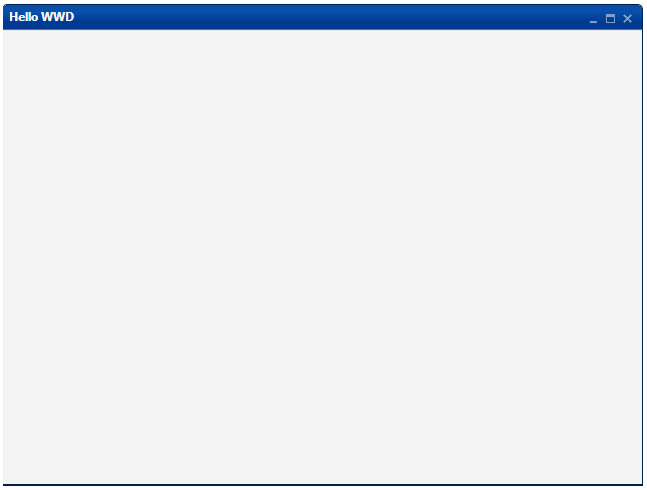
As of release 2.0.0 of the qjide with WWD, 22 different widgets are implemented. All of these have a look-and-feel which is expected for modern desktop applications.

Qooxdoo provides a few widgets more, as well as many more options to set/get on the currently implemented ones. To keep things simple, the most common widget options are usable through WWD. More will come over time, or upon user request.

The following descriptions are always noted as single wwd commands. Please keep in mind, that if you are using multiple wwd commands in sequence, you must use wwdbeg, wwdadd, and wwdend (take a look at the examples).

Window

A window is the container widget for all subsequent widgets.



Create:

```
wwd 'create window <name>' ;< <width>;<height>;'<caption>'
```

Set:

```
wwd 'set <name> visible'   ;< 'true'|'false'|'main'
wwd 'set <name> modal'     ;< 'true'|'false'
wwd 'set <name> resizable' ;< 'true'|'false'
wwd 'set <name> buttons'   ;< 'true'|'false';'true'|'false';'true'|'false'
wwd 'set <name> size'      ;< <width>;<height>
wwd 'set <name> caption'   ;< '<caption>'
wwd 'set <name> status'    ;< 'true'|'false'
wwd 'set <name> text'      ;< '<text>'
```

Event:

```
<name>_appear =: 3 : ...
```

Note:

When setting the window visible with option 'main', the window shows full-screen in the browser, and cannot be closed.

⇨ Useful for user-ready web applications, with no IDE.

When setting buttons, three arguments are needed, all of 'true' = Show or 'false' = Hide:

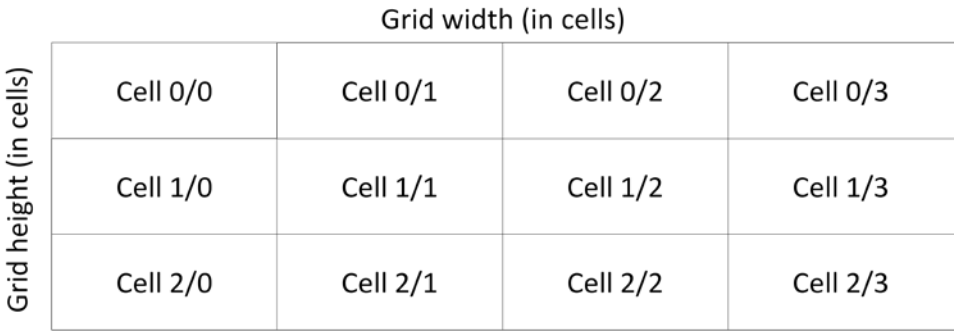
Arg1 = Minimize button in the windows caption bar
Arg2 = Maximize button in the windows caption bar
Arg3 = Close button in the windows caption bar

“status” = ‘true’ or ‘false’ displays a status bar at the bottom of a window.

“text” = ‘status text’ sets the status text.

Grid Layout

The grid layout is the one and only layout item currently supported by WWD. Grid layouts are invisible. They just specify the position and size (number of grid cells) of the widgets.



Create:

```
wwd 'create grid <name> <parent>' ;< <width>;<height>
```

Set:

```
wwd 'set <name> rowspacing'       ;< <size>
wwd 'set <name> colspacing'       ;< <size>
wwd 'set <name> rowalign'         ;< <row>;<hor>;<ver>
wwd 'set <name> colalign'         ;< <col>;<hor>;<ver>
wwd 'set <name> rowmin'           ;< <row>;<minsize>
wwd 'set <name> rowmax'           ;< <row>;<maxsize>
wwd 'set <name> colmin'           ;< <col>;<minsize>
wwd 'set <name> colmax'           ;< <col>;<maxsize>
```

Note:

<hor> may be one of the following values: 'left' or 'center' or 'right'

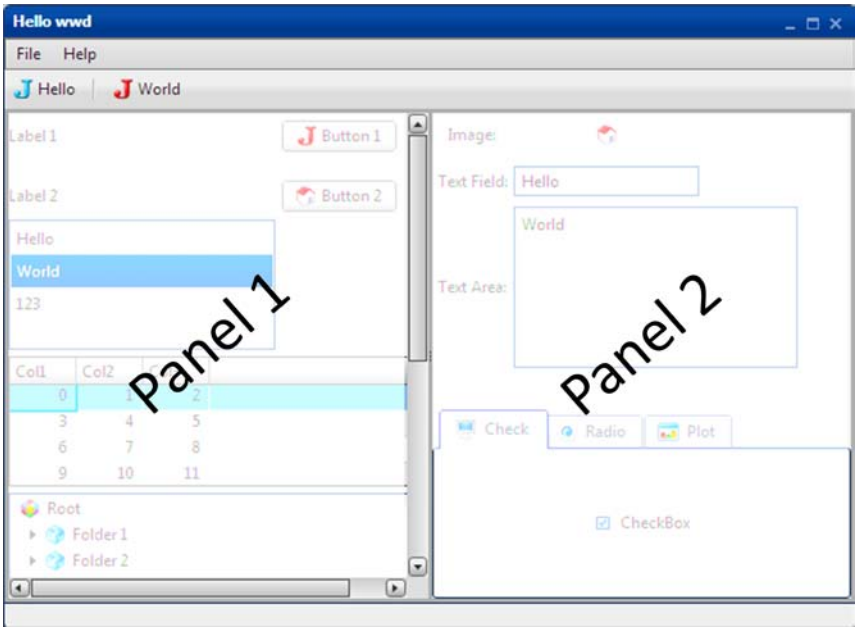
<ver> may be one of the following values: 'top' or 'middle' or 'bottom'

<size>, <minsize>, <maxsize> are always given in pixels.

Splitter

A splitter splits its parent into two separate panels, horizontally or vertically. If a splitter is named “mainsplit” (for example), two panels are automatically created: “mainsplit1” and “mainsplit2”. Each of these panels must have a grid layout assigned, so widgets can be placed inside these panels.

If the panel’s contents are larger than the panel’s size, scroll bars are created automatically.



Create:

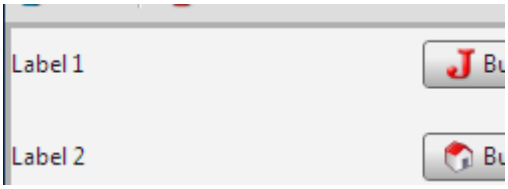
```
wwd 'create splitter <name> <parent>' ;<
   <x>;<y>;<w>;<h>;'horizontal'|'vertical';<size1>;<size2>
```

Set:

```
wwd 'set <name> orientation'        ;< 'horizontal'|'vertical'
wwd 'set <name1>|<name2> visible'   ;< 'true'|'false'
wwd 'set <name1>|<name2> bgcolor'   ;< '#rrggbb'|'reset'
wwd 'set <name1>|<name2> fgcolor'   ;< '#rrggbb'|'reset'
```

Label

A label is a simple text.



Create:

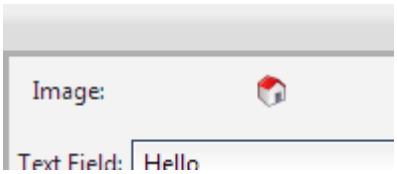
```
wwd 'create label <name> <parent>'  ;< <x>;<y>;<w>;<h>;'<text>'
```

Set:

```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> bgcolor'            ;< '#rrggbb'|'reset'
wwd 'set <name> fgcolor'            ;< '#rrggbb'|'reset'
```

Image

An image shows a bitmap file. Let's say an image label. The image path must be given relative to the qjide's document root: …/qjide/webapp = document root.



Create:

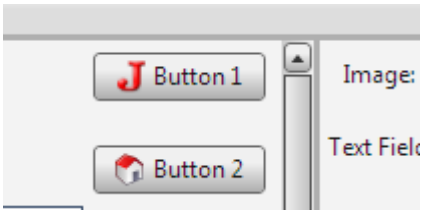
```
wwd 'create image <name> <parent>'  ;< <x>;<y>;<w>;<h>;'<image>'
```

Set:

```
wwd 'set <name> image'              ;< '<image>'
```

Button

This is a simple push button with a text and an optional icon (image). For the icon path, the same rules apply as for the image widget.



Create:

```
wwd 'create button <name> <parent>' ;<
    <x>;<y>;<w>;<h>;'<text>';'<icon>'
```

Set:

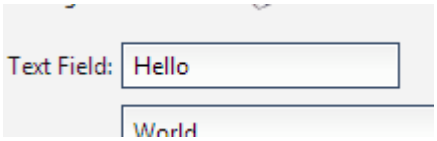
```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> icon'               ;< '<icon>'
```

Event:

```
<name>_execute =: 3 : ...
```

TextField

A text field contains a single line of text.



Create:

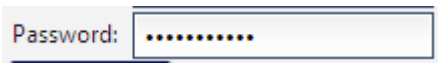
```
wwd 'create field <name> <parent>'  ;< <x>;<y>;<w>;<h>;'<text>'
```

Set:

```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> readonly'           ;< 'true'|'false'
wwd 'set <name> bgcolor'            ;< '#rrggbb'|'reset'
wwd 'set <name> fgcolor'            ;< '#rrggbb'|'reset'
```

PasswordField

A password field contains a single line of text, not shown to the user.



Create:

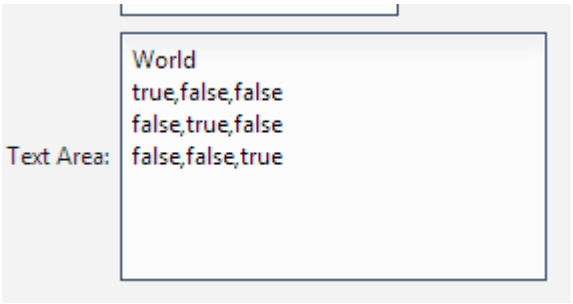
```
wwd 'create pass <name> <parent>'  ;< <x>;<y>;<w>;<h>;'<text>'
```

Set:

```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> readonly'           ;< 'true'|'false'
wwd 'set <name> bgcolor'            ;< '#rrggbb'|'reset'
wwd 'set <name> fgcolor'            ;< '#rrggbb'|'reset'
```

TextArea

A text area can contain many lines of text.



Create:

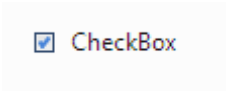
```
wwd 'create area <name> <parent>'   ;< <x>;<y>;<w>;<h>;'<text>'
```

Set:

```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> readonly'           ;< 'true'|'false'
wwd 'set <name> bgcolor'            ;< '#rrggbb'|'reset'
wwd 'set <name> fgcolor'            ;< '#rrggbb'|'reset'
```

CheckBox

A check box is somewhat like a button with a check mark. It can have exactly to states: true or false / checked or unchecked.



Create:

```
wwd 'create check <name> <parent>'  ;< <x>;<y>;<w>;<h>;'<text>'
```

Set:

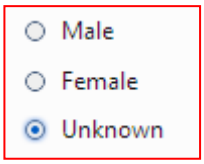
```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> checked'            ;< 'true'|'false'
```

Event:

```
<name>_execute =: 3 : ...
```

RadioBox

A radio box acts as a container for radio buttons. After creating a radio box, assign a grid to it, to layout the radio button widgets. A radio box is the invisible container around the radio buttons.

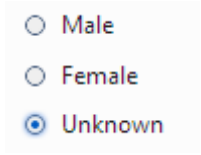


Create:

```
wwd 'create radbox <name> <parent>' ;< <x>;<y>;<w>;<h>
```

RadioButton

A radio button is somewhat similar to a check box, but always only one radio button per radio box, can be the checked (selected) one.



Create:

```
wwd 'create radio <name> <parent>'  ;< '<text>'
```

Set:

```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> checked'            ;< 'true'|'false'
```

Event:

```
<name>_execute =: 3 : ...
```

Spacer

From time to time, it's necessary to insert a dummy (invisible) widget, just to make the layout looking properly.

Create:

```
wwd 'create spacer <name> <parent>' ;< <x>;<y>;<w>;<h>;<wid>;<hei>
```

Note:

<wid> and <hei> are the width and height of the spacer in pixels.

ListBox

A list box (normally) contains many items. Only one of them can be selected.

Create:

```
wwd 'create list <name> <parent>'   ;<
   <x>;<y>;<w>;<h>;(<<data>);<index>
```

Set:

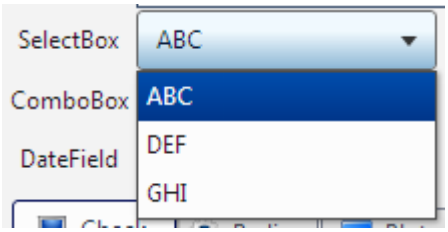
```
wwd 'set <name> data'               ;< <<data>;<index>
wwd 'set <name> selected'           ;< <index>
```

Event:

```
<name>_change =: 3 : ...
```

SelectBox

A select box contains many items. Only one item is shown when collapsed, and only one item can be selected.



Create:

```
wwd 'create select <name> <parent>'  ;< <x>;<y>;<w>;<h>;<<data>
```

Set:

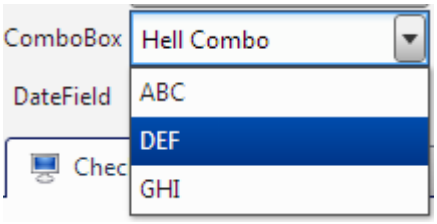
```
wwd 'set <name> data'               ;< <<data>
```

Event:

```
<name>_change =: 3 : ...
```

ComboBox

A combo box acts like a normal text field, but can contain many items to choose from.



Create:

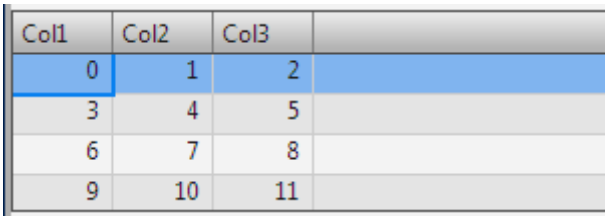
```
wwd 'create combo <name> <parent>'  ;<
   <x>;<y>;<w>;<h>;'<text>';<<data>
```

Set:

```
wwd 'set <name> data'               ;< '<text>';<<data>
```

Table

A table contains (like a list) many items separated into columns. A table is useful to display data tables. Only one row can be selected.



Col1	Col2	Col3
0	1	2
3	4	5
6	7	8
9	10	11

Create:

```
wwd 'create table <name> <parent>'  ;<
   <x>;<y>;<w>;<h>;<<cols>;(<<colwidths>);(<<data>);<index>
```

Set:

```
wwd 'set <name> columns'            ;< <<cols>
wwd 'set <name> data'               ;< <<data>;<index>
wwd 'set <name> selected'           ;< <index>
wwd 'set <name> coltype'            ;< <col>;'string'|'number'
```

Event:

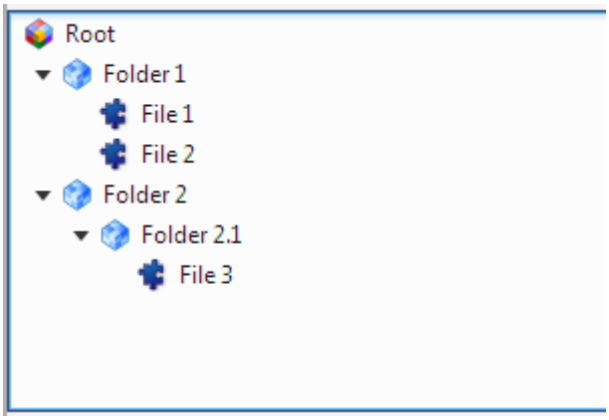
```
<name>_change =: 3 : ...
```

Note:

A column with coltype = 'string' is aligned left.
A column with coltype = 'number' is aligned right.

Tree

A tree shows its item hierarchically. Only one item can be selected.



Create:

```
wwd 'create tree <name> <parent>'   ;< <x>;<y>;<w>;<h>
```

Add:

```
wwd 'add root    <name> <parent>'   ;< '<text>';'<icon>'
wwd 'add folder  <name> <parent>'   ;< '<text>';'<icon>'
wwd 'add file    <name> <parent>'   ;< '<text>';'<icon>'
```

Note:

On creation, <parent> is the layout item (grid owner)

On add folders or files, <parent> is <name> specified for root or other names.

Set:

```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> icon'               ;< '<icon>'
```

Event:

```
<name>_change =: 3 : ...
```

DateChooser

A date chooser months and days. One date per date chooser can be selected.

Create:

```
wwd 'create date <name> <parent>'   ;<
   <x>;<y>;<w>;<h>;<year>;<month>;<day>
```

Set:

```
wwd 'set <name> date'   ;< <year>;<month>;<day>
```

Note:

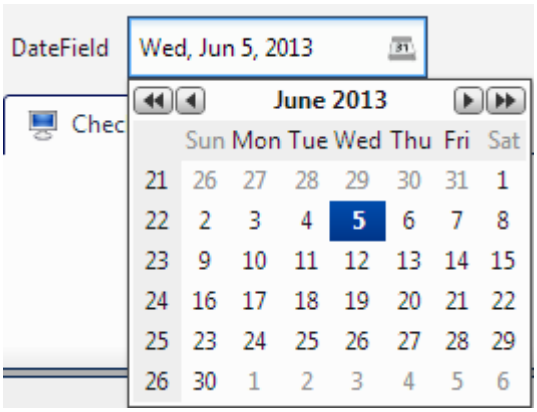
Use <year>=0 <month>=0 <day>=0 to specify the current date.

Event:

```
<name>_change =: 3 : ...
```

DateField

A date field shows a date in various formats. When the selection button is pressed, a date chooser is displayed.



Create:

```
wwd 'create datfie <name> <parent>' ;<
   <x>;<y>;<w>;<h>;<year>;<month>;<day>;'<format>'
```

Set:

```
   wwd 'set <name> date'            ;< <year>;<month>;<day>
   wwd 'set <name> format'          ;< '<format>'
```

Event:

```
   <name>_change =: 3 : ...
```

Note:

Use <year>=0 <month>=0 <day>=0 to specify the current date on the frontend (browser).

Use Unicode date format patterns, see:
http://www.unicode.org/reports/tr35/#Date_Format_Patterns

Example:

EEE, MMM d, yyyy	=>	Thu, Jul 4, 2013
dd.MM.yyyy	=>	04.07.2013

ToolBar

A toolbar is normally located at the top of a window and contains buttons to access frequently used functions of the application.

Create:

```
wwd 'create tbar   <name> <parent>' ;< <x>;<y>;<w>;<h>
```

Add:

```
wwd 'add button    <name> <parent>' ;< '<text>';'<icon>'
wwd 'add separator <name> <parent>' ;< ''
```

Set:

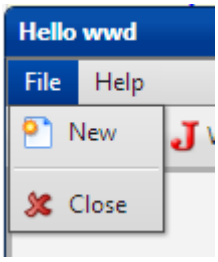
```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> icon'               ;< '<icon>'
```

Event:

```
<name>_execute =: 3 : ...
```

Menu (Pull down Menu)

Pull down menus contains all (or most) of the applications functions.



Create:

```
wwd 'create menu   <name>         ' ;< ''
```

Add:

```
wwd 'add button    <name> <parent>' ;< '<text>';'<icon>'
wwd 'add separator <name> <parent>' ;< ''
```

Set:

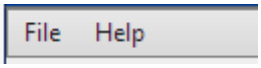
```
wwd 'set <name> text'               ;< '<text>'
wwd 'set <name> icon'               ;< '<icon>'
```

Event:

```
<name>_execute =: 3 : ...
```

MenuBar

The menu bar contains menus (pull down menus) earlier created using: “create menu”.



Create:

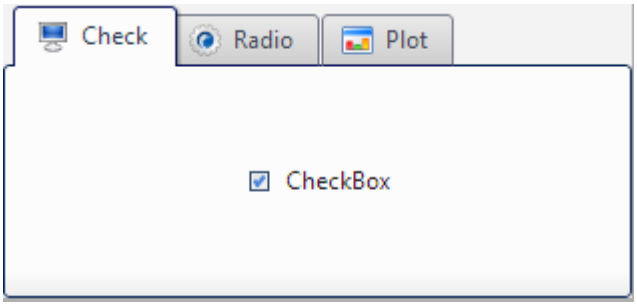
```
wwd 'create mbar   <name> <parent>' ;< <x>;<y>;<w>;<h>
```

Add:

```
wwd 'add    <name> <menu> <parent>' ;< '<text>'
```

TabView

Tabs are useful to switch between multiple pages full of widgets.



Create:

```
wwd 'create tabs   <name> <parent>' ;< <x>;<y>;<w>;<h>
```

Set:

```
wwd 'set           <name> active  ' ;< <page-name>
```

Event:

```
<name>_change =: 3 : ...
```

Add:

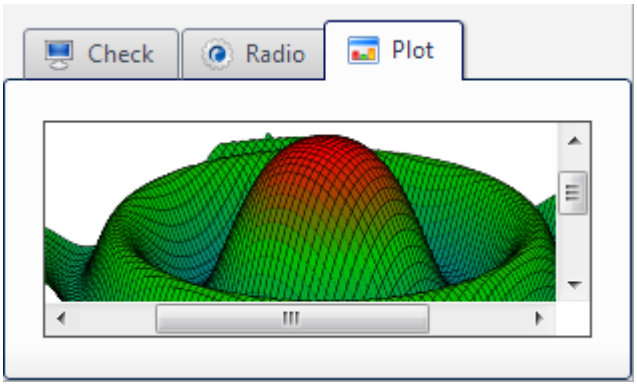
```
wwd 'add    page   <name> <parent>' ;< '<text>';'<icon>'
```

Set:

```
wwd 'set           <name> text    ' ;< '<text>'
wwd 'set           <name> icon    ' ;< '<icon>'
```

Plot

The plot (or viewmat) output from qjide may be displayed in a plot widget. Currently, only one plot widget can be created.



Create:

```
wwd 'create plot   <name> <parent>' ;< <x>;<y>;<w>;<h>
```

Timer

A timer periodically fires an event. It's useful for updating the GUI (see example StockTerm).

Create:

```
wwd 'create timer  <name>         ' ;< <interval>
```

Set:

```
wwd 'set           <name> interval' ;< <interval>
wwd 'set           <name> start   ' ;< ''
wwd 'set           <name> stop    ' ;< ''
```

Event:

```
<name>_execute =: 3 : ...
```

Note:

<interval> is given in milliseconds.

During execution of the timer event handler, other events are disabled. So use timers with care, and do not specify a too short interval.

If the timer interval is to short, the GUI may be become unresponsive.

Event Handlers

Event handlers are normal monadic J verbs. They receive no arguments. An event should always be built as follows:

```
<name>_<event> =: 3 : 0

   <get user interface data using J verb "wwdget">

   <do some tasks>

   <send wwd data back to frontend using J verbs "wwd*">

)
```

Please note:

Sending data to the frontend (web browser) must be the one and only output, generated from an event handler verb.

Do not use “smoutput” for debugging stuff.

How to use the J verb “wwdget”:

When an event if fired by the frontend (web browser) the states and values of all GUI widgets are sent to the backend (web server) and then passed to J. In J, the noun “wwdinfo_qjide_” holds all the information of the frontend GUI. The verb “wwdget” is just a utility to extract values from “wwdinfo_qjide_” in an easy way:

```
<noun> =. <widget-name> wwdget <property>
```

For a text field (named tex1, for example), just do a:

```
tex =. 'tex1' wwdget 'text'
```

Special WWD commands

There are two special wwd commands: “reset”:

```
wwd 'reset'
```

This command resets (clears) all the widgets. It’s useful during development.

and: “delete <name>”:

```
wwd 'delete <name>'
```

This command deletes a single widget. It’s useful for dynamic user interfaces.