

24

Customising FrameMaker 12/13

Overview

Introduction	2
Terminology.....	3
General procedure for customisation.....	5
UI properties	7
Workspaces.....	7
Toolbars	8
Files for the UI.....	9
Relationship between the UI files	10
Menus	11
Customising menus	12
Example menu customisation.....	13
Toolbars.....	14
Customising Toolbars	14
Example toolbar.....	14
Configuration files.....	17
Statements in configuration files	18
Debugging customisation files	19
Tags in toolbar files.....	20
Tool tags.....	21
Detail tag	22
Toolbar commands	23
Toolbar icons	24
Extract icons	26
Commands.....	27
Command examples	27
Command statements	28
Modify statement	29
Command details	30
Some particularities of commands.....	34
Command names from plugins and scripts	38
Menu statements	42
Workspace definition	45
Initialisation files maker.ini and others	47

Introduction

From the beginning FrameMaker could be customised.

- Menus can be switched from complete to quick. This is a standard feature.
- Menus can be modified.
- Commands can be created and referenced in menus and toolbars.

Until the arrival of the new user interface with FM-9 the process was described in an Adobe file, which was lastly issued for FM-7. The text provided by this file was grosso modo still valid for FM 11. However, the new interface requires significantly more information, which is not available from Adobe until now.

Document history

This document is not sacrosanct. In particular it has not been verified by Adobe and hence may contain errors or misinterpretations. Readers are requested to communicate errors or enhancements to the author klaus(daube.ch)

- 2013-02 Use FM-11 version of document as a base. FM-12-M1 introduces two button sizes and colours. Image reference in *toolbar.xml* files can use a base-name. The additional product interface XML-Author has no influence on this document.
- 2014-05 Clarify the situation with non-existing toolbar-icons.
- 2014-09 Check the situation with FM-13-M1. It seems that nothing related to the customisation has changed.
- 2015-03 Reworking the structure of the document in some places to get a ‘more logical’ sequence.
- 2015-12 Add text concerning scripts as commands. Amend the example toolbar with this.
- 2016.01 Change the document title to reflect scope of document; rework structure; add scheme and examples.
- 2016-02 Clarifications concerning command names from scripts and plugins, clean up the example files.
- 2016-12 “Shortcuts in ExtendScript” added.

Sources

- Experiments and beta testing activities.
- Communication in FrameMaker related forums as well as personal communication with members of these forums.
- (Very sparse) communication with Adobe developers.
- User guides for FrameMaker.
- [Adobe blogs](#)
- Video: [Getting started with the new FM-9 UI](#)
- Video: [Workspace overview](#)
- Video: [Customize and manage the Workspace](#)

Explanation of fonts and highlights

Menus and names from the user interface are **in > this > font**. Variables (place holders) use *italic* script. Keywords and the like are in fixed pitch font.

This is sample code

Terminology

Note: *IMHO the terminology of FM has not yet settled. The term pod should be eliminated - but is used since FM-9 and confuses more than it clarifies. Panel, palette and dialogue are somewhat synonymous. Terminology is not coherent between the workspace files, the tool tips and context menus.*

For better understanding of some terms look at the picture [Elements of the workspace](#) on page 5

\$HOME The FM installation directory. In my case this is
H:\Adobe\FrameMaker.12en\AdobeFrameMaker12 - and
H:\Adobe\FrameMaker.13en\AdobeFrameMaker2015¹⁾.

Access character An access character²⁾ in a menu item is preceded by the '&' symbol (e.g. E&xit).

In Windows 7 and beyond these access characters are visible at the menu items only when pressing the left **ALT** key. To use the short cut for Exit, you type left ALT f x.

Dialogue Prior to FM-9 a dialogue was either modal or non modal. Some of these have been replaced by dockable panels which can be grouped into panel groups and minimised to icons.

Even in FM-12 there are still many of the old dialogues active.

Dock A dock is an area on the application window to which UI elements can be docked (anchored) and aligned.

Dock content	What can be in
palatte toolbar	Iconised or expanded dialogs/pds, the graphics palette
multi-controlbar	Horizontal groups of toolbars on top of the application window
palette	Special tab panes such as the bottom pod (palette-kit data)

FCode The function code is the connection between the command (as defined in a configuration file such as `cmd.cfg`) and the routine in the application which performs the function.

Menus and commands The UI information basically is split into 2 groups:

- Command sets assembled according to views.
- Menus and tool bars assemble according to views.

modal vs. non-modal

- A modal dialogue must be closed before work can continue on the document.
- A non-modal dialogue could stay open like a palette.

Panel A docked or undocked dialogue, which need not be closed to work on the document. The automatic behaviour depends on **Preferences > General > Interface > Pods**.

Panel group A collection of panels which can be handled as a whole.

Pod A dockable panel. This seem to be a synonym for panel. The term was first used in RoboHelp.

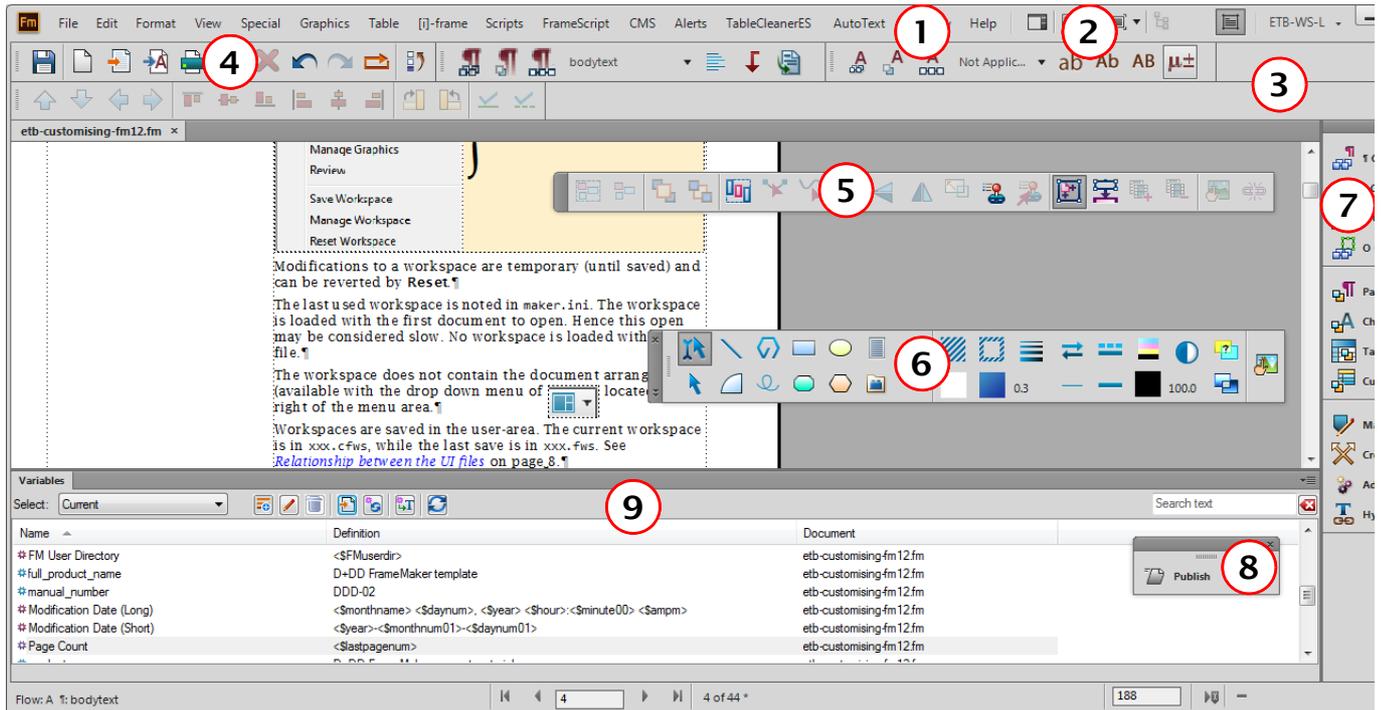
1 The installation program does not allow to modify the last level. This is due to new mechanism (introduced with FM-10) using a data base for installation/de-installation.

2 The Adobe document calls this a *mnemonic shortcut*. The access character must be carefully chosen to avoid duplicates within a menu.

UI	User interface. The elements of user interaction: windows, panes, menus and dialogues. Also keyboard shortcuts belong into this category.
Palette	This term was first used in FM for the read-only FM-documents which behave as non modal dialogues (Equation palette, Vertical toolbar, Template browser, Element catalogue ...). This term is still used in the workspace files.
Product interface	FM \geq 12 contains three product interfaces (also known as modes): Structured, UnStructured and XMLAuthor. XMLAuthor is a stripped down version of the structured interface and also available as own product.
Toolbars	Toolbars are an alternative to menus for a desired function.
User-area	In Windows 7 this is C:\Users\user\AppData\Roaming\Adobe\FrameMaker\11\
View	Views were introduced with FM-11. A view groups elements of a workspace. Hence there are more menus and toolbar groups than before FM-11. In Structured mode there are three views: XMLView, AuthorView, and WYSIWYGView. See Relationship between the UI files on page 10. These are represented as icons in the top right hand corner before the Workspace selection:
	
	Unstructured mode knows only the WYSIWYGView. In the above picture this is the rightmost icon (active). This has consequences for the customisation: it is no more sufficient to have a \$HOME\fm\init\configui\cstomui.cfg file. The contents of such a file must also be appended to a menu file.
Workspace	A workspace is a saved set of frequently used panels/toolbars in a desired arrangement for repeated use.

Elements of the workspace

The workspace contains various elements, which are demonstrated on the following screen shot:



- 1 Menu bar
- 2 Arrangement panel. A chosen arrangement (e.g. 2 document panes side by side) is not saved in the workspace file.
- 3 Multi control bar (dock) spanning multiple rows of tool bars. In the workspace file a row of this is a Control Bar Pane.
- 4 Toolbar, docked. In the workspace file this is a Control Bar.
- 5 Toolbar, undocked = floating
- 6 Graphic toolbar (undocked, horizontal arrangement)
- 7 Tab group (docked) with minimised panels.
- 8 Tab group (undocked)
- 9 Bottom pod - a special palette.

If used, work space information (workspace, tool bars, menus), are copied from \$HOME to the user area. Any modifications are only kept there.

For details see [Relationship between the UI files](#) on page 10.

General procedure for customisation

If a requirement can not be satisfied with entries in the [Initialisation files maker.ini and others](#) on page 47, consider setting up a custom workspace:

- 1 For special menu entries set up a custom menu file by means of `customui.cfg` and append it to the relevant menus.

- 2 A custom toolbar requires creation of a `custom-toolbar.xml` file and a modification of the toolbar set file `fmtoolbar.xml`. It may require the creation of button images.
- 3 For both cases it may be necessary to define new commands in `customui.cfg` (which must be appended to the relevant menus).
- 4 To define a custom workspace copy an existing workspace file (`xxx.fws`) to a custom named file and modify the references to menu and toolbar set.

Download example customisation

You can download the example customisation files from [my website](#).

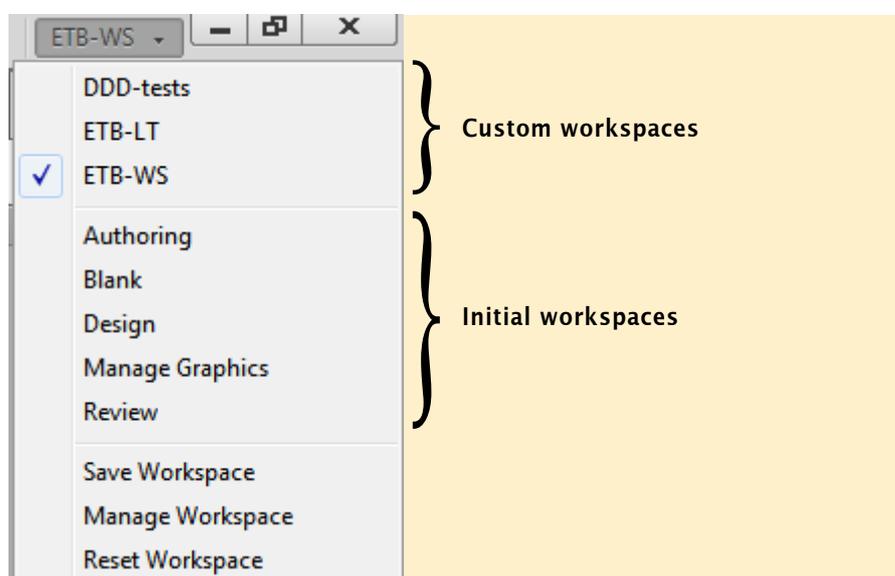
UI properties

Workspaces

A workspace is a saved set of frequently used panels/toolbars in a desired arrangement for repeated use. It also offers flexibility of screen usage, by allowing a user to place panels in numerous possible forms/arrangements: default, iconic, minimized, docked (left, right, bottom, top), floating, grouped.

The workspace remembers the dialogues that were open at the previous session and also their positions, size, state etc.

FrameMaker ships with a set of standard Workspaces tailored for different tasks. They can be modified and then saved with a new name. FM also provides an empty workspace to start with.



Modifications to a workspace are temporary (until saved) and can be reversed by **Reset**.

The last used workspace is noted in `maker.ini`. The workspace is loaded with the first document to open. This open may be considered slow. No workspace is loaded with a book file.

The workspace does not contain the arrangement of documents (available with the drop down menu of  located to the right of the menu area).

Workspaces are saved in the user-area. The current workspace is in `xxx.cfws`, while the last saved is in `xxx.fws`. See [Relationship between the UI files](#) on page 10.

At the first use of workspaces the necessary files (workspace definition, toolbars, menus) are taken from `$HOME` and copied to the user-area.

To design a new workspace for a specific task, open all the required panels and save the Workspace using **Save Workspace**. The names are case sensitive.

Toolbars

- Tool-set** The toolbars available for the UI are listed in an xml file which is referenced in the workspace file (see [Relationship between the UI files](#) on page 10). The standard name is `fmttoolbar.xml`³. There may be more toolbar files, but only those listed in this file are visible in the menu.

fmttoolbar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<FMTOOLBARLIST version="1">
  <TOOLBAR file="graphics.xml"/>
  <SEPARATOR sep_id="0"/>
  <TOOLBAR file="quick_access.xml"/>
  <TOOLBAR file="structured.xml"/>
  <SEPARATOR sep_id="1"/>
  <TOOLBAR file="text_format.xml"/>
  <TOOLBAR file="table_format.xml"/>
  <TOOLBAR file="para_format.xml"/>
  <SEPARATOR sep_id="2"/>
  <TOOLBAR file="align_object.xml"/>
  <TOOLBAR file="object_properties.xml"/>
  <SEPARATOR sep_id="3"/>
  <TOOLBAR file="trackchanges.xml"/>
</FMTOOLBARLIST>
```

- Toolbar** The toolbars themselves are defined in xml files. The only toolbar which can not be customised is the graphics tool palette. Toolbars can be docked in the top toolbar pane, to the right or left of the application window or undocked (floating).

Note: *Toolbars containing a drop-down list can not be docked to the left or right!*

³ A much clearer term would be `toolbar-set.xml`

Files for the UI

If used, work spaces are copied to the user area. Any modifications are kept only there.

FM-12 and later contains three interfaces: Structured, UnStructured and XMLAuthor. XMLAuthor is a stripped down version of the structured interface and also available as own product. It is not elaborated further here.

The UI information basically is split into 2 groups:

- Command sets assembled according to views
- Menus and tool bars assemblies according to views.

Product interface	Structured			Unstructured
	AuthorView	CodeView (XML)	WYSIWYG	WYSIWYG
View				
Standard commands	Commands independet of views are located in \$HOME\fminit\configui: cmds.cfg			
Commands ^a	mathcmds.cfg wincmds.cfg	wincmds.cfg	mathcmds.cfg wincmds.cfg	mathcmds.cfg wincmds.cfg
Work spaces ^b	Authoring.fws none.fws ^c	Authoring.fws none.fws	Authoring.fws Blank.fws Design.fws Manage Graphics.fws none.fws Review.fws Structured Authoring...	Authoring.fws Blank.fws Design.fws Manage Graphics.fws none.fwsReview.fws
Menus	menus.cfg	menus.cfg	menus.cfg menus_review.cfg menus_structured_au- thoring.cfg menus_ts.cfg ^d	menus.cfg menus_review.cfg menus_ts.cfg
Tool bars	fmtoolbar.xml ^e graphics.xml quick_access.xml quick_element.xml structured.xml trackchanges.xml	fmtoolbar.xml quick_access.xml xpathtoolbar.xml xslttoolbar.xml	align_object.xml fmtoolbar.xml fmtoolbar_review.xml graphics.xml object_properties.xml para_format.xml quick_access.xml quick_access_review. xml structured.xml table_format.xml tag-description.xml text_format.xml trackchanges.xml	align_object.xml fmtoolbar.xml fmtoolbar_review.xml graphics.xml object_properties.xml para_format.xml quick_access.xml quick_access_review. xml structured.xml table_format.xml tag-description.xml text_format.xml trackchanges.xml
Customisation	\$HOME\ fminit\configue\cusomui.cfg This file contains both menu defintions and (hypertext) command defintions.			

- a. Commands are located in fminit\configui\interface\view\
- b. Work spaces are located in fminit\WorkSpaces\interface\view\
- c. none.fws is an empty workspace used if no document or book is open. It can be used to build a custom work space from scratch. It can refer to a custom menu, but does not honour the definition of a custom toolbar set — IMHO this is an error.
- d. The purpose of menus named menus_ts.cfg is IMHO unclear.
- e. fmtoolbar.xml does not define commands and icons for a tool bar, but lists all the tool bars available in this work space.

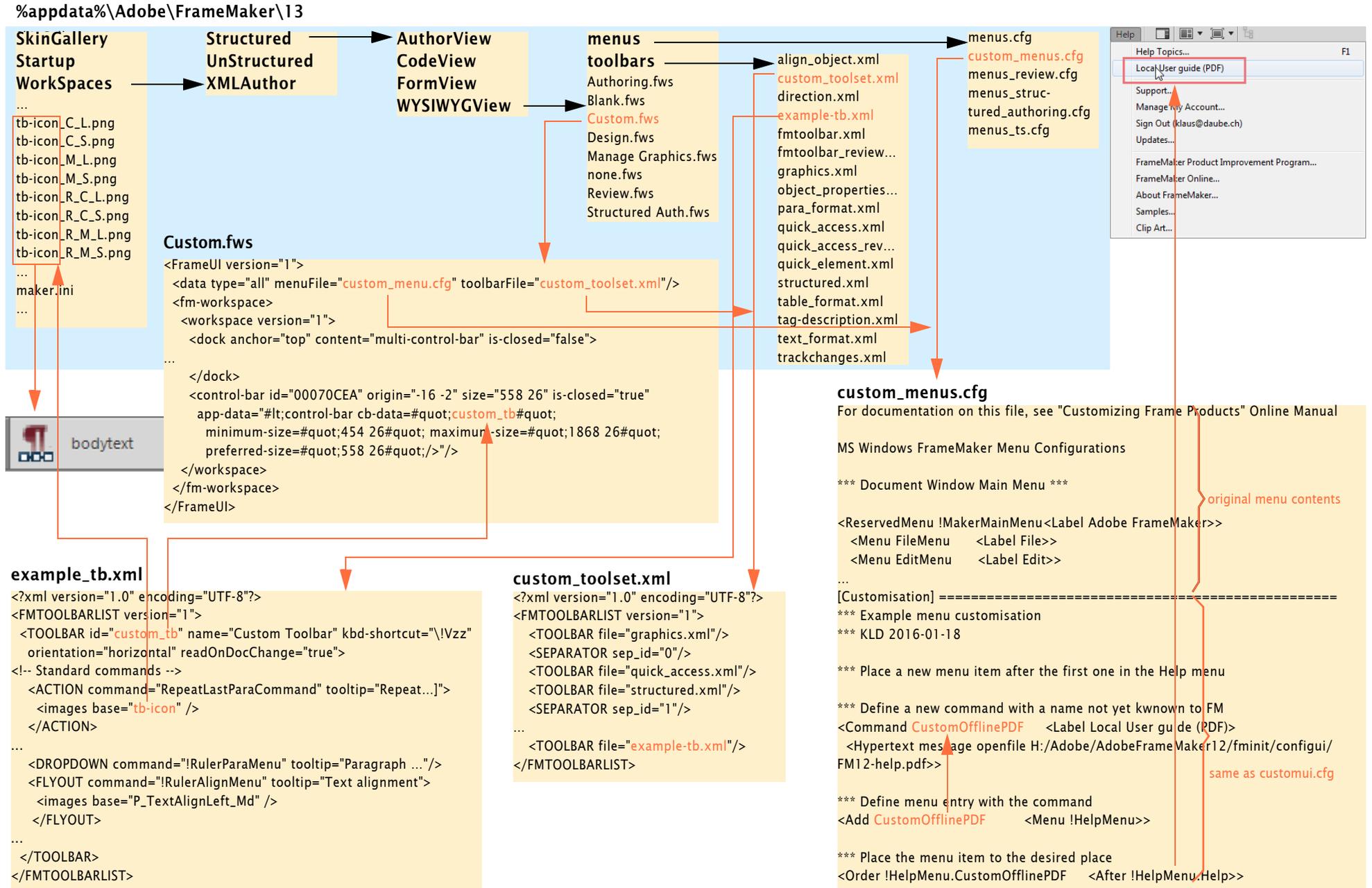
2016-12-03

E:_DDDprojects\FM-toolbar00\AllIETB\etb-customising-fm12.fm

LD+D D

Relationship between the UI files

24-10



Menus

customui.cfg For FM versions prior to FM-11 all customisation was established in the file `fminit\configui\customui.cfg`.
The possibilities in this file are explained extensively in the Adobe document *Customisation of Frame Products (FM-7)*.
This file is still required if the menu modifications are to be visible from the beginning.⁴⁾
However, the contents of this file must also be appended to the relevant menu file. It is good practice to copy a menu file to a new file, e.g. `custom-menu.cfg` and then append the contents of `customui.cfg` to it.

Views Since FM-11 the contents of menus depend on a view. Hence there are several menu files. See *Files for the UI* on page 9. As an example the WYSIWYG view of the unstructured interface of FM-12/13 is listed:

```
$HOME\fminit\configui\
  cmds.cfg
  sample.cfg
  UnStructured\
    WYSIWYGView
      mathcmds.cfg
      wincmds.cfg

$HOME\fminit\WorkSpaces
  UnStructured
    WYSIWYGView\
      Authoring.fws
      Blank.fws
      Design.fws
      Manage Graphics.fws
      Review.fws
      menus\
        menus.cfg
        menus_review.cfg
      toolbars\
        align_object.xml
        fmttoolbar.xml
        fmttoolbar_review.xml
        graphics.xml
        object_properties.xml
        para_format.xml
        quick_access.xml
        quick_access_review.xml
        structured.xml
        table_format.xml
        tag-description.xml
        text_format.xml
        trackchanges.xml
```

- 4 `customui.cfg` in FM-12 and later is only relevant before a workspace is selected - that is, before any book or documents have been opened. After closing all documents or books no customisation is active. `none.fws` is an empty workspace. It can be used to build a custom work space from scratch. It can refer to a custom menu, but does not honour the definition of a custom toolbar set — IMHO this is an error.

Customising menus

Customisation

To create your custom menu, take the appropriate standard menu and copy it to a new file. Name this file with a prefix of your customisation project. For example `custom-menus.cfg`.

You develop the customisation in a file `customui.cfg` which is located in `$HOME\fm\init\configui\`. As long as you have not a customised menu assigned to a workspace, the menu customisation is only visible during the display of the splash screen (before you have opened a document or book).

To be available regularly the *contents* of `customui.cfg` must also be appended to a menu file, which is referenced in the workspace (see [Relationship between the UI files](#) on page 10).

To be available also with no document or book open, workspace `$HOME\fm\init\WorkSpaces\UnStructured\none.cws` must point to the modified menu file (see table footnote c on [page 9](#)):

```
<FrameUI version="1">
  <data type="all" menuFile="custom-menus.cfg" toolbarFile="fmtoolbar.xml"/>
  <fm-workspace>
```

Testing menus

You do not need to restart FM to test a modified menu. With **View > Menus > Modify...** it can be read into the current workspace.

Errors in the customisation file are reported in the FM console window. Even if an error is found, reading and interpreting the file continues.

Commands in menus

In the various existing menus you can find a desired command for your special menu entry. You may however need to set up a new command:

- Open a file. For example the offline user guide pdf.
- Execute a program.
- Execute a FrameScript
- Execute an ExtendScript

The first two tasks are performed by hypertext commands:

Open a file

```
Hypertext message openfile H:/Adobe/AdobeFrameMaker12/fm\init\configui\FM12-help.pdf
```

Execute a program

```
<Command ETBfmConsole <Label Console Log File>
  <Hypertext message openfile H:/Adobe/AdobeFrameMaker12/
  fm\init\configui/GetLogFile.exe>>
```

Hyper-links with relative paths always point to files in the same directory as the parent file. While for documents this can be interpreted, it is not clear what it means for menus.

Relative paths do not start in `$HOME` as in pre-FM9. They start in the current document folder:

`$path[initdir]/fm\init` does not work (neither upper/lower/mixed case, nor with path in quotes or double quotes).

For addressing scripts see [Note concerning scripts](#) on page 23.

Execute a FrameScript

```
<Add iFrameMultiCatalog <Menu FormatMenu>>
```

Execute an ExtendScript

```
<Add ImportFormatsSpecial <Menu FormatMenu>>
```

Example menu customisation

Let's assume that you want to modify the **Help** menu.

You are tired of searching for the offline user guide (pdf). Once you have downloaded it via **Help > Help Topics > Getting Started > General Resources**. You placed it in `$HOME\fminit\configui\` with the name `FM-12-help.pdf`.

You want this in the menu as **Help > User Guide PDF**.

We will establish this only for the unstructured interface in the WYSIWYGView.

Note: *This example is part of the download of [FM-customisation-example.zip](#).*

customui.cfg

In a plain vanilla FM installation this file does not exist (or it is empty). Hence you need to create it. This file must be in Windows code page (cp 1252), not in UTF-8⁵!

```
*** Example menu customisation
*** KLD 2016-01-18

*** Define a new command with a name not yet known to FM
<Command CustomOfflinePDF      <Label Local User guide (PDF)>
  <Hypertext message openfile H:/Adobe/AdobeFrameMaker12/fminit/configui/FM12-help.pdf>>
*** Define menu entry with the command
<Add CustomOfflinePDF          <Menu !HelpMenu>>
*** Place the menu item to the desired place
<Order !HelpMenu.CustomOfflinePDF  <After !HelpMenu.Help>>
```

Note: *For test purposes you may reference any pdf file in the command. Adjust the file location - use forward slash (/) in place of backslash (\). Only absolute paths work!*

Modify the relevant menu

The customisation by `customui.cfg` is only active when no document or book is open⁶.

In our case the menu to be modified is `menus.cfg`. The file contents of `customui.cfg` must be appended to the relevant menu of the workspace to be available for open documents or books.

Since it is not a good idea to just modify a standard menu, we copy it to `menus_custom.cfg` and then add the contents of `customui.cfg` to this file.

Define a new workspace

Our modified `menus_custom.cfg` must be referenced in a workspace. Hence Create a copy of `Authoring.fws` in the directory `Unstructured\WYSIWYGView` and name it `Custom.fws` Edit line 2 according to the following (modification highlighted):

```
<data type="all" menuFile="menus_custom.cfg" toolbarFile="fmtoolbar.xml"/>
```

After starting FM and selecting the Custom Workspace drop-down list, the menu modification is active.

- 5 The file coding is not at all consistent:
cp 1252: *.cfg (`customui.cfg`, `cmd.cfg`, `mathcmd.cfg`, `menu.cfg` etc.)
UTF_8: *.xml (toolbars), *.fws, *.cfws (work space), `maker.ini`.
- 6 Since FM-12 the rôle of `customui.cfg` is unclear. See footnote 4

Toolbars

Customising Toolbars

Customising toolbars or creating new toolbars is more elaborate than customising menus, because more files are involved (See [Relationship between the UI files](#) on page 10):

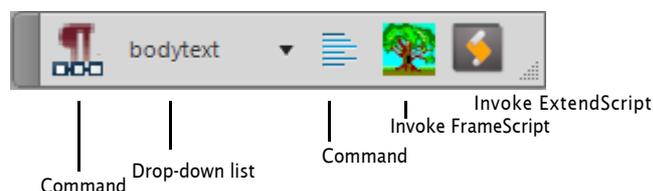
- The toolbar file itself (e.g. custom-tb.xml).
- The list of toolbars in fmtoolbars.xml.
- The icons needed for the toolbar.
- Probably a workspace file Custom.fws to refer to the customised tool bar (and probably also customised menu).

Note: *Toolbars containing a drop-down list can not be docked at left or at right to become oriented vertically!*

Example toolbar

Note: *This example is part of the download of [FM-customisation-example.zip](#).*

The following is an example toolbar with all types of widgets. We want to integrate it into the workspace Custom.fws already used for the menu customisation.



This tool bar shall be available in the unstructured interface, in the WYSIWYGView of the Custom workspace.

Note: *If not specially noted, the file locations start with %appdata%\Adobe\FrameMaker\vv\ with vv being 12 or 13.*

Establish an example tool bar

The following ingredients are required:

- File example-toolbar.xml defining the buttons and their commands as well as keyboard-short cuts. This file must be put into all the relevant view folders, in our case into ...WorkSpaces\UnStructured\WYSIWYGView\toolbars\
- A copy of fmtoolbar.xml with modifications giving file custom_toolset.xml for all the relevant view folders (in our case, same as mentioned before).
- An appropriate set of icons. Either you reference existing icons in the dll-files or you provide them in the user area. After start of FrameMaker select the Authoring View, for which you have set up the two files custom_toolset.xml and example_tb.xml.

In menu **View > Toolbars** you will find the toolbar and can activate it.

Move the tool bar to any location or dock it. You may save the current workspace with a new name.

You may also update the workspace Custom containing already the menu customisation.

File example-tb.xml

This file describes the contents of the toolbar. This example references standard commands as well as a FrameScript and an ExtendScript.

These files are available in the already mentioned download.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <FMTOOLBARLIST version="1">
3   <TOOLBAR id="custom_tb" name="Custom Toolbar" kbd-shortcut="\!Vzz"
4     orientation="horizontal">
5     <!-- Standard command -->
6     <ACTION command="RepeatLastParaCommand" tooltip="Repeat last ¶ command [Esc j j]">
7       <images base="etb-para-repeat" />
8     </ACTION>
9     <!-- Drop-down List -->
10    <DROPDOWN command="!RulerParaMenu" tooltip="Paragraph formats"/>
11    <!-- Fly-out menu -->
12    <FLYOUT command="!RulerAlignMenu" tooltip="Text alignment">
13      <images base="P_TextAlignLeft_Md" />
14    </FLYOUT>
15    <!-- simple FrameScript -->
16    <ACTION command="MyCommand1" tooltip="Just an alert">
17      <images base="tst-fs" />
18    </ACTION>
19    <!-- simple ExtendScript -->
20    <ACTION command="msgJS" tooltip="Message via JavaScript">
21      <images base="tst-estk" />
22    </ACTION>
23  </TOOLBAR>
24 </FMTOOLBARLIST>

```

The ID of the toolbar need not be the same as the file name. I choose custom_tb for the ID, although the file names is example_tb.xml. The ID is used in the workspace files only.

For each tool button a type is defined:

- ACTION Button for standard command.
- DROPDOWN Open a drop down list of commands.
- FLYOUT Open a sub menu of commands.

See [Tags in toolbar files](#) on page 20 for more details.

Toolbar commands

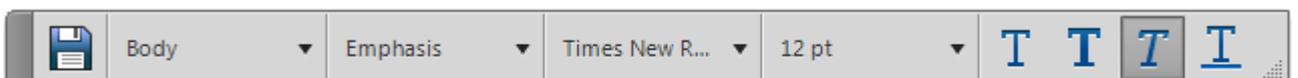
When setting up a toolbar you need to have an idea which commands to use. Get the exact names from the appropriate menus (see [Configuration files](#) on page 17) or from a command list. See for example www.daube.ch.

Note: *Hypertext commands can only be used in menus, not for toolbars.*

For more details see [Commands](#) on page 27.

Note on drop-down lists

More than one drop-down list can appear in a tool bar⁷⁾:



⁷ Example provided by Shmuel Wolfson.

Toolbar icons

Standard icons	These (e.g. P_TextAlignLeft_Md) are located in fmres.dll or owlres.dll of FrameMaker. Names of these can be deduced from existing tool bars or from my web site (look for “tool bar resources”).
Custom icons	Those named etb-xxx and tst-xxx are custom icons which must reside in the user area (no subdirectory possible).

File custom_toolset.xml

- 1 Make a copy of fmtoolbar.xml and name it custom-toolset.xml
- 2 Modify this file as shown hereafter.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <FMTOOLBARLIST version="1">
3   <TOOLBAR file="graphics.xml"/>
4   <SEPARATOR sep_id="0"/>
5   <TOOLBAR file="quick_access.xml"/>
6   <TOOLBAR file="structured.xml"/>
7   <SEPARATOR sep_id="1"/>
8   <TOOLBAR file="text_format.xml"/>
9   <TOOLBAR file="table_format.xml"/>
10  <TOOLBAR file="para_format.xml"/>
11  <SEPARATOR sep_id="2"/>
12  <TOOLBAR file="align_object.xml"/>
13  <TOOLBAR file="object_properties.xml"/>
14  <SEPARATOR sep_id="3"/>
15  <TOOLBAR file="trackchanges.xml"/>
16  <SEPARATOR sep_id="4"/>
17  <TOOLBAR file="direction.xml"/>
18  <SEPARATOR sep_id="5"/>
19  <TOOLBAR file="example_tb.xml"/>
20 </FMTOOLBARLIST>

```

Integrating toolbar into workspace

The toolbar will appear in menu **View > Toolbars**. There you activate it and it will initially float around.

The workspace mechanism has assumed a certain width of the toolbar which can be adjusted with the lower resize handle:



As usual the toolbar is docked to the other toolbars with the docking handle.

You can now save the current workspace, with or without a different name.

Custom workspace

You may also update the Custom workspace as defined in [Customising menus](#) on page 12 and modify it to directly use the custom tool set:

```
<data type="all" menuFile="menus_custom.cfg" toolbarFile="Custom_toolset.xml"/>
```

Configuration files

Configuration files define both commands and menus. These files still have the format as in previous FM versions. That is, they are named xxx.cfg and use the well-known MIF syntax (not xml).

The names of these configuration files are specified in the initialisation file for FrameMaker (see [Files] in maker.ini).

customui.cfg The file fminit\configui\customui.cfg is a special configuration file called customisation file, because with this file commands and menus are customised.

If this file does not exist or does not have the name defined in maker.ini (see [Files] in maker.ini) then no customisation is performed.

Commands The commands are defined in three files located in \$HOME\fminit\configui:

File	Contents
cmds.cfg	General commands, Escape sequences
mathcmds.cfg	Commands for the Equation Editor
wincmds.cfg	Platform dependent commands, definition of shortcuts

With the introduction of views, multiple files cmds and wincmds. exist.

Menus Until FM-10 menus were located in \$HOME\fminit\maker. FM-11 introduced the concept of views which requires a multitude of menus. See *Relationship between the UI files* on page 10.

Note: *While standard menus are in their own files (separated from commands) the customisation file customui.cfg may contain both commands and menus. Also a custom menu may contain both kinds of definitions.*

Configuration file statements

A configuration file consists of a series of statements that define menus, menu items, and the order of those items. Commands are also defined in configuration files and may contain definitions for details which may also be present in menu files.

Properties of statements

- Statements are case-sensitive.
- Each statement is enclosed in angle brackets (< and >).
- Statements must appear in a particular order.
- A statement begins with a keyword defining its function.
- A statement may span several lines.
- Text outside angle bracket-pairs is treated as comment. Don't include angle brackets in comments. Hence if you want to comment out statements, you need to replace the < > symbols, for example, by { }.

comment examples

```
[etb --- ETB addenda] =====
*** The [label] supports file navigation in EditPad
    - This file (customui.cfg) can not be UTF-8, it must
      use Windows encoding and FrameRoman coding for Labels
    - Default path ($HOME) is named here + fm-root+
      (no blank after first +). This is exchanged by the
      installation pgm with the real $HOME directory.
```

Initialisation sequence

When FrameMaker starts, it first reads the standard menu and command configuration files and then a customisation file⁸). The information in each file overrides the information in files read previously. Hence the following order is necessary:

- 1 Definition of the commands.
- 2 Modification of labels, shortcuts.
- 3 Definition of a menu item referring to a command.
- 4 Order of the menu item or sub menu within parent menu.

Statements in configuration files

Purpose	Statement, statement detail
Define a command	Command on page 28
Define a command for a menu item that is chosen while the Shift key is held down	ShiftCommand on page 28
Define a new label for a command or menu item	Modify on page 29
Define the function to be called when a command is chosen	Definition on page 30
Define a label for a menu or command that is visible in the user interface	Label on page 30
Define a context-sensitive label for a menu or menu item.	ReservedLabel on page 30
Define a keyboard shortcut for a command	KeySequence on page 31
Define a label for the shortcut which appears next to the command name on the menu	KeySequenceLabel on page 32
Define whether a command is a general command, a FrameMath command (for the Equation Editor), or both	Mode on page 33
Define an Asian typography command	AsianFonts on page 33
Define a new menu	Menu on page 42
Define a new reserved menu	Reserved menu on page 42
Add a menu item to a menu	Add on page 43
Define a particular place for a menu item on a menu.	Order on page 44
Remove a menu or menu item	Remove on page 28

⁸ This process is called *Localisation* in the progress indication during the start. This is due to the fact that the menu files are different for each UI language. In FM the UI language is defined at installation - it can not be changed afterwards.

Debugging customisation files

If you're writing a lengthy menu customisation file, consider writing and testing the customisations a few at a time. This will make it much easier to locate problems in the statements you write. As you create the file, you can save the file and then read it into FrameMaker to test your statements.

With **View > Menu > Modify...** the modified menu can be read.

To display error messages when you load a menu customisation file, set `ShowErrors` in `maker.ini` to **On**. You can also turn **On** the keyboard shortcut alerts (`ConfigWarnKbdOverride`, `ConfigWarnKbdOverride`) to see error messages in the console window⁹. If you find errors, you can fix them immediately and continue writing.

When you read the same menu customisation file again¹⁰, you'll see error messages about redefining a command (because the same statements are being read again). Ignore these messages. Use comments throughout the menu customisation file to document your work. Others may need to edit the file later.

-
- 9 Please note that even a plain vanilla FrameMaker installation will create a huge number of such messages, because definitions are 'overloaded'. It may be difficult to find real errors this way ...
 - 10 I'm not certain whether this method with **View > Menus > Modify...** works reliably in FM-11 and beyond. We have now at least two modification files...

Tags in toolbar files

Note: Be aware that in strings (e.g. defining the tool tip) no & must be used. Character entities (e.g. #amp;) are not resolved. Use the word 'and' or the '+' sign in place of the '&'.

First line of toolbar file	<pre><?xml version="1.0" encoding="UTF-8"?></pre>
Comments	Comments are in the standard XML/HTML format: <pre><!-- One line comment --> <!-- Multiline comment intermediate line last line --></pre>
	Note: XML comments must not appear before the FMTOOLBARLIST statement!
Example	See File example-tb.xml on page 15. It displays most of the following tags.

FMTOOLBARLIST

This is the root tag in a toolbar file.

Syntax	<pre><FMTOOLBARLIST attributes /> TOOLBAR statement with details </FMTOOLBARLIST></pre>
Attributes	version="1" Parser version for which this toolbar file was written. Value needs to be compatible with the current parser version.

TOOLBAR

This tag defines the toolbar and is a wrapper for the items on the toolbar.

Syntax	<pre><TOOLBAR attributes /> tool tags (ACTION, DROPDOWN, ...) </TOOLBAR></pre>
Attributes	<p>file File name of the toolbar, if the description is to be picked from somewhere else. If this attribute exists no other attributes are parsed here. This attribute is not present in any of the Adobe toolbars up to FM-13.</p> <p>id Unique identifier for the toolbar, for workspace identification, FDK access and API notifications</p> <p>name Name of the toolbar as visible in the menu. This may contain character entities, for example name = "Paragraphs #amp; Characters"</p> <p>orientation Keyword defining the default orientation of the toolbar: horizontal: Default orientation is horizontal. vertical-narrow: Default orientation is vertical and the items are arranged in a single column. vertical-wide: Default orientation is vertical and the items are arranged in two columns.</p> <p>dock Keyword defining the preferred/default dock (currently horizontal toolbars can be docked only at the top):</p>

left:	Toolbar will be docked at the left anchor
right:	Toolbar will be docked at the left anchor
top:	Toolbar will be docked at the top anchor
none:	Toolbar will be floating

kbd-shortcut Key-sequence to activate the toolbar (default = none)

readOnDocChange I have no idea, what this is good for. Found only in quick_element.xml.

Tool tags

ACTION

The action tag defines the command assigned to an UI button.

Syntax `<ACTION attribute/>`
`<IMAGES ...>`
`</ACTION>`

<i>Attributes</i>	command	Identifier of an already defined FM action - a required attribute.
	name	Name of the action (default is the tag defined for the command)
	tooltip	Tool tip displayed on mouse hover (default is the name of the action)
	help	Help String for the action command (default= none)
	image	An image-name for the action (default= None) ¹¹ .

TOGGLE

A toggle tag is used to define two (logically alternating) actions to be performed from a single widget.

Note: *I have not managed to get this working. I have also not seen this tag in any Adobe tool bar yet.*

Syntax `<TOGGLE attributes1 />`
`<command attributes2 />`
`</TOGGLE>`

<i>Attributes1</i>	name	Name of the toggle (default= None)
	tooltip	Tool-tip that is displayed on mouse hover (default= name of the toggle)
	help	Help String for the toggle command (default= None)
	image	An image-name for the toggle (default= None) ¹¹ .
<i>Attributes2</i>	on	Required identifier of an already defined FM action (non-toggle type).
	off	Required identifier of an already defined FM action (non-toggle type).

¹¹ This tag attribute may be replaced by `images = list of images` (see [IMAGES](#) on page 22)

FLYOUT

This is used to define a popup menu.

Syntax	<code><FLYOUT attributes /></code>	
Attributes	command	Identifier of an already defined FM Menu - a required attribute.
	name="	Name of the flyout (default= the tag defined for the command)
	tooltip	Tool tip that is displayed on mouse hover (default= name of the flyout)
	help	Help string for the flyout command (default= none)
	image	An image-name for the flyout (default= none) ¹¹).

DROPDOWN

This is used to (generally) define a menu whose sub items are a list of options that can be chosen one at a time, for example, fonts.

Syntax	<code><DROPDOWN attributes /></code>	
Attributes	command	Identifier of an already defined FM Menu - required field"
	tooltip	Tool tip that is displayed on mouse hover (default= name of the drop-down)
	help	Help String for the drop-down command (default= none)

Note: *There is no width indication for this widget. The width is assumed by the workspace mechanism. If more than one drop-down list appears in a tool bar then resizing ins applied to all proportionally.*

SEPARATOR

This tag places a separator between two items

```
<SEPARATOR/>
```

Detail tag

Detail tags are optional. Currently only the `IMAGE` tag is in this category.

IMAGES

This tag describes the images displayed on an `ACTION`, `FLY-OUT` and `TOGGLE`. Alternatively only one image can be defined (see [ACTION](#) on page 21).

Syntax	<code><images attributes /></code>	
Attributes	normal	Default image displayed when the UI is bright (default= none)
	rollover	Image displayed on mouse hover when the UI is bright (default= normal image)
	dark_normal	Default image displayed when the UI is dark (default= normal image)
	dark_rollover	Image displayed on mouse hover when the UI is dark (default= dark_normal image)

Example See [Using icon-images](#) on page 24.

Toolbar commands

See also section [Commands](#) on page 27.

All the menu items/commands that end up executing an FCode can be used here. Hence hypertext commands can not be used. However, scripts (FrameScript, ExtendScript) can be referenced. See [Example toolbar](#) on page 14.

Note concerning scripts

- Scripts must be loaded during the start of FrameMaker. This action generates a command name and an FCode for the script.
- The FCode assigned to a script may differ from FM session to FM session. Hence it is useless.
- **ExtendScripts** are located in
%appdata%\Adobe\FrameMaker\vv\Startup\
The name of the script name (without the file extension) provides the command name. Installed ExtendScripts get command names from their internal command definitions.
- **FramExtendScripts** may be located anywhere. However it must be assured that they are loaded at start of FrameMaker. See FrameScript options.
- The name for the command may not be identical to the script file name (excluding the file extension) due to an initial script. You may get the proper command name by the free FrameScript [Report FM Commands](#) from itl.

File name of script	Command name
ImportFormatsSpecial.jsx	ImportFormatsSpecial
MultiCatalog.fso	iFrameMultiCatalog
RemoveUnusedFormats.fsl	ESLSSRUN305 ^a

a. This is an automatically generated command name.

For details see [Command names from plugins and scripts](#) on page 38.

Toolbar icons

- Standard icons** Icons for toolbars and for the dialogues are located in resource files: `fmcustom.dll`, `fmres.dll`, `owlres.dll`, `fmdlgl.dll` and are not relevant in this context.
- Names of such icons can be deducted from existing tool bars or from my [web site](#) (look for “tool bar resources”).
- Custom icons** If images are needed, which are not in these resources, they must be of type png (Portable Network Graphic) and be located in the user-area (in the same directory as `maker.ini`).
- Hovered (roll over) icons are framed. When clicked the icon it gets a darker background. Hence the icons must be transparent.

Large and coloured icons

Since FM-12 both large and/or coloured icons are supported. Regular sized icons are 18×18 pixels, large icons are 26×26 pixels. Hence the maximum number of icon images is 8.

You can have 8 image files for the following possible combinations of preferences with base name e.g. `xyz`:

Preferences		Icon names	
Size	Colour	Normal	Rollover
Size	Colour	Normal	Roll over
Large (L)	Coloured (C)	<code>xyz_C_L.png</code>	<code>xyz_R_C_L.png</code>
	Greyscale (M)	<code>xyz_M_L.png</code>	<code>xyz_R_M_L.png</code>
Regular (S)	Coloured	<code>xyz_C_S.png</code>	<code>xyz_R_C_S.png</code>
	Greyscale (M)	<code>xyz_M_S.png</code>	<code>xyz_R_M_S.png</code>

In most cases the `R_X_X` forms are copies of the `_X_X` forms.

Using icon-images

- 1 Add the base name of the icon in the base attribute of the image element.

```
<ACTION command="RepeatLastParaCommand"
  tooltip="Repeat last ¶ command [F4]">
  <images base="etb-para-repeat" />
</ACTION>
```

- 2 Create at least 2 icon images for normal and rollover state of the icon. For example if the icon name is `xyz` the image names will be `xyz_C_S.png` and `xyz_R_C_S.png`. (Here, C= colour, S=regular, and R=rollover.) If, however, your icon preferences are set to have large or greyscale icons instead of regular and colour, you will use M and L in the icon names.
- 3 You can further add more icon files for icon states, such as `dark_normal` and `dark_rollover` by specifying attributes with data in the relevant element. For example:

```
<ACTION command="CenterPara">
  <images base="P_TextAlignCenter_Md"
    dark_normal="<icon_name>.png" <!-- regular size -->
    dark_rollover="<icon_name>.png"
    dark_normal_l="<icon_name>.png" <!-- large size -->
    dark_rollover_l="<icon_name>.png"
  />
/>
```

Example icons These are all from fmcustom.dll, base name = P_AddRows

S - Regular (18 × 18)				L - Large (26 × 26)			
C - Coloured		M - Grey		C - Coloured		M - Grey	
Normal	Roll over	Normal	Roll over	Normal	Roll over	Normal	Roll over
_C_S	_R_C_S	_M_S	_R_M_S	_C_L	_R_C_L	_M_L	_R_M_L

- Although there are special images for the roll-over situation, the pictures are the same (_C_S = R_C_S).
- In roll-over state the buttons are surrounded by a border and the background becomes lighter than in normal state. Hence there is no need for special images.

Default icons

The result of missing images depend on the ‘severity’ of the problem:

If size does not fit, the defaults are boxes with text in them:



If set to regular size, standard icons not found are depicted with a default icon:



Tool bar icons and workspaces

- A workspace defines the toolbars to be active in it.
- References to all sorts of icons (small/large, grey/coloured) exist in the tool bar.
- The size and colour of icon to be displayed is only defined in **Preferences > Global > Interface > Icons**.

If you do not see coloured icons after switching to a workspace which should have them, check your settings in **Preferences**. After changing these preferences a restart of FrameMaker is required.

Contents of the resource files in fminit

With the new UI two additional resource files were introduced: owlres.dll and owlres.res. These contain png images which are not handled by (to me) known resource editors. Also fmcustom.dll now contains png images, no more bit maps (bmp).

File	Icons, pictures	Dialogues	Other
fmcustom.dll	Images for toolbar also icons for the panels and graphic toolbar. 2 to 8 variants per image.	none	version info
fmdlg.dll	Rubi-bit maps [bmp]. Many new items (panels)	classic dialogues, panels, panel-lists	Icngroup (icons for the dialogues) version info; 500 (?)
fmres.dll	Button images for dialogues, palettes and panels (16x16), [bmp]. Some items no more used	none	Cursor group (32x32 cursors); Icngroup (icons in panels etc.)
owlres.dll	Images for the new toolbars [png]. 2 to 4 variants per image. This file is new with FM-9	C-like definitions for application bar ^a , grafix bar, UI preference dialogue etc.	Xstr (strings with all text in xml notation); version info (correct)

a. This relate to functions to the right of the menu bar (UI visibility, Arrange documents, Screen mode)

Extract icons

To get the icons out of the DLLs - to have individual files to work on for the tool bar buttons - my procedure is the following:

Extract and rename

- 1 Open the dll in **ResHacker**.
 - Select the appropriate resource type (e.g. bitmap).
 - Save the resources with **Action > Save [...] resources**.
 - Use an rc name such as fmres-bmp.rc .
- 2 Open the rc file in **EditPad**.
 - Convert double to single line spacing
 - Use the REGEX to exchange the 'columns':

fmcustom	binaries	(.+)	png	"(.+)"	\2\t\1
fmres	bitmaps	(.+)	bitmap	"(.+)"	\2\t\1
	cursors	(.+)	cursor	"(.+)"	\2\t\1
	icons	(.+)	icon	"(.+)"	\2\t\1
owlres	binaries	(.+)	png	"(.+)"	\2\t\1

- Remove blanks at start of line
 - Save the table as fmres-rename-table-bmp.txt
- 3 Start **RenameByTable.ahk**
 - Fill in all fields, including the target file extension.

PDF of all icons

- 1 Open the directory (e.g. E:\FM-specials\FM-12-tests\Resources-owlres\renamed-png) in **Thumbs+**.
- 2 In **Image > Print Catalog** set up a layout (or use icon-overview) with the following properties:
 - Printer = Adobe PDF
 - Print Thumbnail borders OFF
 - Colour output
 - Margins all: 0.5cm
 - Thumbs width 4.5cm, height 2.2cm
 - Header: Resource icons xxx
 - Header font 12 pt, Caption font 8pt
 - Items for caption: only File name
 - Files to process: Current folder
 - Print Heading for each folder: OFF
 - FINISH: provide file path for PDF file

Be aware that some file names are too long for complete display. In the tabular arrangement of the icons the names may overlap.

Commands

There are three command files in \$HOME\fminit\configui:

```
cmds.cfg
mathcmds.cfg
wincmds.cfg
```

In the various views there are different versions of these files, since not all commands are relevant in a particular view.

For customisation it is not necessary to modify any of the standard command files. All customisation of commands is done in customui.cfg. and one or more menu files.

Note: *A complete list of FM-13 commands can be found on www.daube.ch. These command lists are created with the free FrameScript [Report FM Commands](#) from itl.*

Command examples

For explanation of the keywords see [Command statements](#) on page 28.

Normal command	<pre><Command NewDocument <Label Document...> <KeySequence \!fn> <Definition \x300> <Mode All>></pre>	<p>Name of the command</p> <p>What you see in the menu</p> <p>Shortcut (ESC sequence)</p> <p>FCODE, the command definition</p> <p>Valid contexts for this command</p>
Command with restricted context	<pre><Command SelectAll <ReservedLabel Flow Select All in Flow> <ReservedLabel Frame Select All in Frame> <ReservedLabel Page Select All on Page> <KeySequence \!ea> <Definition \x327> <Mode All>></pre>	
Modify the shortcut	<pre><Modify SelectAll <KeySequence ^a> ></pre>	
... and indicate it in the menu	<pre><Modify SelectAll <KeySequenceLabel CTRL+A> ></pre>	
Combine these two modifications	<pre><Modify SelectAll <KeySequence ^a> <KeySequenceLabel CTRL+A >></pre>	

2016-12-03

E:_DDDprojects\FM-toolbar00\AllIETB\etb-customising-fm12.fm

L D+D D

Command statements

Command

The command statement is the wrapper definition for the command:

Syntax `<Command cmd-name <detail1> <detail2> <detailN>>`

Details may be added to a command also by the [Modify statement](#) on page 29.

Cmd-name A unique name of the command. This serves as a reference between the various statement types.

Details For the detail specifications see [Command details](#) on page 30.

Examples See also [Command examples](#) on page 27.

```
<Command PrintingDisplay
  <ReservedLabel Document &Printing Display>
  <KeySequence \!qqp >
  <KeySeqLabel Esc q q p>
  <Definition \x4F1 \x4F2 \x4F3 \x3F8>
  <Mode All>>
```

Note: *This command is defined by 4 function codes which imposes some problems. See the remark at [Multi-code commands](#) on page 35.*

Custom command A custom command must not use an already existing name. Hence it is good practice to prefix the name with an indicator, for example:

```
<Command ETBVertToolBar ...>      Enhanced Toolbar
<Command _MTCharSet ...>         Microtype's Customisation
```

ShiftCommand

This statement defines a command for a menu item that is chosen while the Shift key is held down. This statement normally appears in a menu file (not a command file).

Syntax `<ShiftCommand cmd-unshifted cmd-shifted>>`

Cmd-unshifted This is the identifier of the command as it normally appears.

Cmd-shifted This is the identifier of the command you want to appear when you hold down the Shift key.

Examples `<ShiftCommand Save SaveAll>`
`<ShiftCommand FindNext FindPrevious>`

Note: ¹²⁾*Commands defined by ShiftCommand can not be placed in context menus. The insertion of the command (by Add) does not create an error, although the command is not inserted. A further Order command will not find the (not) inserted command and create an error. Example creating the error:*

```
<ShiftCommand GraphicsObjProps GraphicsPickObjProps> ...
<Add GraphicsPickObjProps <Menu !GraphicsContextMenu>>
<Order !GraphicsContextMenu.GraphicsPickObjProps
  <After !GraphicsContextMenu.GraphicsObjProps>>
```

Remove

You can not remove commands. Only the menu entry is removed.

Syntax `<Remove cmd-name <Menu menu-id>>`

Examples `<Remove GraphicsReshape <Menu GraphicsMenu>>`
`<Remove GraphicsReshape <Menu QuickGraphicsMenu>>`

¹² I have reported this as bug # 3494702 as of 2013-02-01.

Modify statement

Modify

The Modify statement is used to change details of a command. This command must be defined already. The change may affect:

- the label(s)
- the key sequence aka shortcut(s)
- the key sequence label(s)

Syntax `<Modify cmd-name <new-detail1> ... <new-detailN>>`
`<Modify cmd-name [context-id] <new detail>>`

Cmd name This is the name (ID) of the command whose properties shall be modified

Details In the Modify statement the same details can be defined as in a Command statement. See [Command details](#) on page 30.

Modifications are cumulative for key sequences (shortcuts)¹³. The other details are overwritten by the newest one.

Examples `<Modify NewDocument`
`<KeySeqLabel Ctrl+N>>`
`<Modify TerminateMaker`
`<Label E&xit>> Define Label with access character`

Renaming a context sensitive command To rename the label of a context sensitive command, both the command-name and the context-identifier (here: Frame) must be given:

`<Modify SelectAll <ReservedLabel Frame Select Everything in Frame>>`

Various labels for same command To get a different label for a command in only one place, define a new command that duplicates the function of the old one (using the same key sequence, definition, and mode), but use a different label. Then put the new command on the menu in place of the old one.

¹³ If a customisation file contains shortcut definitions for commands that already have shortcuts defined for them, warning messages may be written to the console log file. This happens with `ConfigWarnKbdRedundant = On` in `maker.ini` (see [\[Preferences\] in maker.ini](#)).

Command details

Definition

This defines the function of the command.

Syntax `<Definition Fcode1 [Fcode2 ... Fcoden]>`

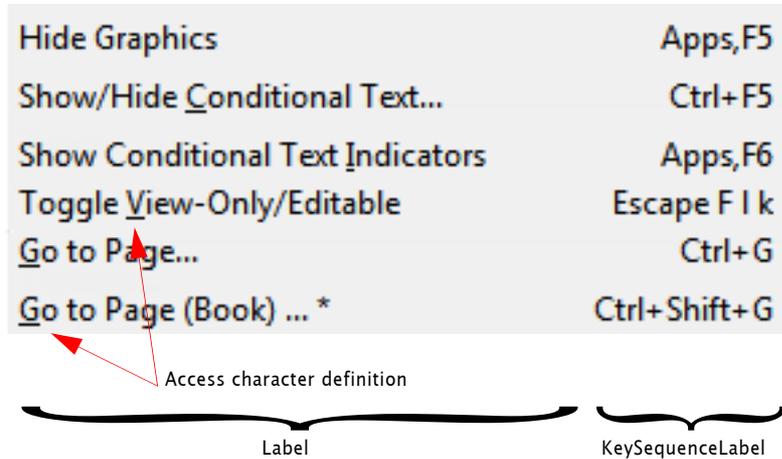
Fcode The function code is the connection between the command and the routine in the application which performs the function. A command may issue several functions, although most commands have only one Fcode associated.

The Fcode is noted as `\xnnn` with `nnn` being a hexadecimal number. You can find relevant Fcodes in the FDK¹⁴⁾ documentation or in the command files or in special lists derived from these files.

Examples `<Definition \x300>` Create new document
`<Definition \x302>` Command Help
`<Definition \x3F1 \x3F2 >` Borders and Text symbols On

Label

The label defines the entry in a menu. It also provides a default for tool tips on buttons using this command.



Syntax `<Label label-string defining the menu entry>`

Label string If an ampersand character (&) is needed in the *label-string*, it must be doubled. This is due to the fact that the & precedes an access character. This will be underlined in the menu.

Example `<Label P&rint Setup...>`

This will display in a menu as **Print Setup...**

ReservedLabel

Some commands have a different label, and a different effect, depending on the context - where the insertion point is, what is selected, and so on. In these cases, the command gets a context-id defining the condition in which it can be chosen. Each of the conditions has a `ReservedLabel` statement.

Syntax `<ReservedLabel context-id label-string>`

Context-id The following context-ids are used in commands¹⁵⁾:

Context-id	Condition
Body	Body pages are displayed
Book	Book window is active

14 In the FrameMaker Developer Kit (FDK) see `include\fcodes.h`

15 This table is deducted from the various `cmdxxx.cfg` files of FM-13. See also `FP_EnabledWhen value` in the FDK reference.

Context-id	Condition
Ditamap	Ditamap is active
Document	Document is active
Flow	Flow is selected
Frame	Frame is selected
Generic	Set up any generated file
History	History window
Long, Long2	Complete (Long) menus is active
LongMultiple	Multiple book components are selected
LongSingle	Single book component is selected
MacEdition...	Deprecated, since Macintosh is no more supported.
Master	Master pages are displayed
NoDelete	This page can not be deleted (e.g. Left/Right master page)
NoName	This page has no name (not yet saved)
NotRegistered	Product not yet registered
Other	Other than body pages are displayed
Page	A document page is active
Redo	Undo command list
Reference	Reference pages are displayed
Registered	Product is registered
Repeat	Repeat xxx
Scratch	Probably a left-over from program development
Search	AFAIK only used to search references.
Short, Short2	Short (Quick) menus is active
Straddle	Selected cells are not straddled
Table	Table is selected
TextInset	Text inset is active
TOC, LOF, ...	Set up the respective generated file TOC, LOF, LOT, LOP, LOE, APL, AEL, LOM, AML, LOR, IX, AIX, SIX, IOM, IOR)
ToTable	Selection is paragraph(s)
ToText	Selection is a table
Undo	Redo command list
Unstraddle	Selected cells are straddled

Label string The same rules as with the Label detail apply. See [Label](#) on page 30

Example

```
<Command SelectAll
  <ReservedLabel Flow Select All in Flow>
  <ReservedLabel Frame Select All in Frame>
  <ReservedLabel Page Select All on Page>
... >>
```

The SelectAll command acts as a place holder on the menu for the group of commands: Flow, Frame, and Page, all of which have the same command definition.

KeySequence

The key sequence defines a keyboard shortcut for the command.

Syntax `<KeySequence sequence>`

Sequence This defines the sequence of keystrokes. Two cases must be distinguished: Keys to be pressed together (key group) and keys to be pressed one after the other (key sequence).

If you need the modifier key symbols literally in a sequence, the symbol must be preceded by a solidus (/). For example to use the '+' in a key sequence literally, you provide '/+'.

Key group A base key is pressed together with modifier keys. These modifier keys are defined with special symbols:

Escape key: \!
 Shift key: +
 Control key: ^
 Alt key¹⁶⁾: ~

A key group should not use more than two modifier keys, because humans have only two hands ...

Key Sequence These sequences mostly start with the Escape key. Non-alphanumeric keys need a special notation¹⁷⁾:

Escape key: \!
 Function keys: /F1 ... /F12
 Insert key: /Insert

Note: *Defining a key sequence must consider existing key sequences. If a key sequence ESC,q,q is already defined, ESC,q can no more be defined, because the input process waits for the next q. If You then enter just any other character (e.g. z), nothing will happen.*

*Existing key sequences mostly mimic the menu entries they support. They depend on the UI language. For example the sequence for **Repeat last character modification** is Esc,c,c in the English, and Esc,z,w in the German FrameMaker.*

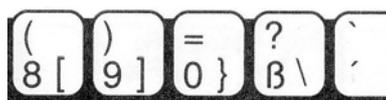
Examples <KeySequence +^b > Shift+CTRL+B (one key stroke)
 <KeySequence \!/+c > Escape, Plus, c (three distinct key strokes)

Shortcuts on the Equations palette The shortcuts that appear on the Equations palette can not be customised. This palette is actually a special view-only document containing hypertext commands. The same is true for the Vertical Toolbar.

KeySequenceLabel

The key sequence label is added to the right side of a menu entry.

Syntax <KeySeqLabel *descriptive-string*>



16 On European keyboards the right **Alt** key is engraved **AltGr** (Alternate Graphic). Pressing this key together with another key types the special graphic engraved at lower right of the symbol key. **AltGr** is equivalent to **Alt+Ctrl**.

17 Since Windows 95 the following keys available only on Windows Keyboards can no more be used for short cuts in FrameMaker:
 Application Key - also called Menu Key (/Apps) right of the space bar. Its standard function is to open the context menu of an application.
 The Windows logo key (/Win) left of the space bar is used for various Windows functions, for example, to open the Start Menu. See [Wikipedia](#).

Descriptive string This echoes the key sequence of the command. You should distinguish the two cases of key sequences by different notation:

Key group Key names concatenated with a '+' sign.
 Key sequence A list of key-names (separated by blank or comma).

Examples <KeySeqLabel CTRL+A> Key group
 <KeySeqLabel Escape, q, q, e> Key sequence
 <KeySeqLabel Esc q q p> Key sequence
 <KeySeqLabel ALT+Shift+F9> Key group

Note: *If a command does not have a KeySeqLabel detail, the command will be displayed on the menu with no shortcut. This does not mean the shortcut does not exist; it just means the shortcut is not displayed on the menu.*

Mode

The Mode defines the validity of a command for a particular environment.

Syntax <Mode mode >

Mode Defined modes (used in commands) are:

Mode	
All	The default
Math	During Equation editor
NonMath	Anything but Math

Example <Mode NonMath >

To define a command that appears in menus only if your system supports typing Asian text in documents and dialogue boxes, use this detail:

Syntax <AsianFonts Yes>

Note: *AsianFonts No has the same effect as omitting the statement. In this case the command applies to all configurations.*

AsianFonts

Some particularities of commands

Hypertext commands

See a synopsis of the hypertext commands at www.daube.ch.

- Hypertext commands can only be used in menus, not for toolbars.
- It is not possible to define shortcuts for hypertext commands. For example, the shortcut defined in the Modify statement is not executed:

```
<Command Test1 <Label Using FrameMaker 11>
  <Hypertext message openfile H:/.../etbfm11-help.pdf>>
<Modify Test1 <KeySequence \!qqq>
  <KeySeqLabel Escape q q q>>
```

- Hypertext commands do not work in books, neither do shortcuts to hypertext commands in a book window. Although it is possible to define an entry in a book menu for such commands, they are inactive (greyed out).
- It is possible to define FM read only documents containing buttons with hypertext commands. Examples are the Equations palette and the Vertical tool bar. See more on this method on www.daube.ch.

Keyboard shortcuts

For ordinary commands (not hypertext commands) keyboard shortcuts can be redefined. However, this will provoke a log message if `ConfigWarnKbdOverride = On` is set in `maker.ini`. Even an unmodified FM-UI will issue such messages¹⁸.

If the same short cut is given more than once to a command, another message is issued if `ConfigWarnKbdRedundant = On` is set in `maker.ini`¹⁹. This happens in particular for settings both in `configui\customui.cfg` and in the menu file in `%appdata%` - which is common for many UI modifications.

Command overloading

A command (name) can be redefined²⁰.

Initial definition

```
<Command Test2
  <Label Using FrameMaker 11>
  <Hypertext message URL
http://help.adobe.com/en_US/.../11.0/Using/index.html>>
<Modify Test2
  <KeySequence \!qqq>
  <KeySeqLabel Escape q q q>>
```

Later definition

```
<Command Test2
  <Hypertext alert Test2 is testing hypertext> >
<Add Test2 <Menu ViewMenu>>
```

Result

The shortcut ESCqqq brings up an alert that says “Test2 is testing hypertext”.

18 For example: The shortcut !ph in file `FMPublisher` overrides one or more previous shortcuts.

19 For example The shortcut !SFL in file `$HOME\fm\init\configui\structured\wysiwygview\wincmds.cfg` is a duplicate of an existing shortcut.

20 Lynne Price, 2014-10-23

Multi-code commands

For multi-code commands only two of the three UI elements may be defined: button, menu item, shortcut. If all three are defined, then only the first code will be executed.²¹⁾

For the following no menu item will be defined:

```
<Command EditingDisplay
  <ReservedLabel Document &Editing Display>
  <KeySequence \!qqe >
  <KeySeqLabel Esc q q e>
  <Definition \x3F1 \x3F2 >
  <Mode All>>
```

And for the following only a menu item (and no shortcut or button) will be defined:

```
<Command ETBGoToMasterPage
  <Definition \x343 \x345 >
  <Label Go to a Specific Master Page... >>
<Add ETBGoToMasterPage <Menu ViewMenu>>
  <Order ViewMenu.ETBGoToMasterPage
  <Before ViewMenu.ViewMasterPages>>
```

OWL commands

The new interface requires special commands:

Show next tab in panel group

```
<Command ShowNextKit
  <Label Show Next>
  <KeySequence ^ /F6>
  <Definition \x971>
  <Mode All>>
```

Show previous tab in panel group

```
<Command ShowPrevKit
  <Label Show Previous>
  <KeySequence ^ +/F6>
  <Definition \x972>
  <Mode All>>
```

???

```
<Command ThemeLoad
  <Label LoadTheme...>
  <KeySequence \!LT>
  <Definition \x974>
  <Mode All>>
```

???

```
<Command ThemeSave
  <Label SaveTheme...>
  <KeySequence \!ST>
  <Definition \x975>
  <Mode All>>
```

Toggle between the following 3 screen modes

```
<Command ToggleScreenMode
  <Label Toggle Screen Mode>
  <KeySequence ~+/Return>
  <KeySequence \!SMt>
  <Definition \x978>
  <Mode All>>
```

Standard mode (UI, normal size)

```
<Command ScreenModeStandard
  <Label Standard Screen Mode>
  <KeySequence \!SMs>
  <Definition \x979>
  <Mode All>>
```

Full width with UI

```
<Command ScreenModeFullWithUi
  <Label Full Screen Mode with UI>
  <KeySequence \!SMu>
  <Definition \x97A>
```

²¹ This is a problem since FM-9 and is most likely a consequence of the new user interface.

Full screen, no UI	<pre><Mode All>> <Command ScreenModeFullScreen <Label Full Screen Mode> <KeySequence \!SMf> <Definition \x97B> <Mode All>></pre>
Preferences > Interface	<pre><Command UiPreferences <Label Interface...> <KeySequence \!ip> <Definition \x980> <Mode All>></pre>
Preferences > Alert Strings	<pre><Command UiAlertStringsPreferences <Label Alert Strings...> <KeySequence \!asp> <Definition \x981> <Mode All>></pre>
Show all tool bars	<pre><Command ToolBarShowAll <Label Show All> <KeySequence \!TSA> <Definition \x989> <Mode All>></pre>
Hide all tool bars	<pre><Command ToolBarHideAll <Label Hide All> <KeySequence \!THA> <Definition \x98A> <Mode All>></pre>

Some graphic commands

Operate on first selected graphic object in a group	<pre><Command SetFirstPenPattern <Label Set First Pen Pattern (black)> <KeySequence \!0p></pre>
Operate on selected graphic object	<pre><Command IncrementPenPattern <Label Increment Pen Pattern> <KeySequence \!\+p></pre>
Keep next selected graphic tool active	<pre><Command GraphicsKeepTool <Label Keep Tool> <KeySequence \!gk></pre> <p>This keeps the next selected graphic tool active until SmartSelectionTool is activated.</p>
Operate on hot spot graphic	<pre><Command GraphicsCreateLink <Label Create Link to graphic...> <KeySequence \!gcl></pre>
Operate on hot spot graphic	<pre><Command GraphicsCreateLinkTable <Label Create link table for graphic...> <KeySequence \!gct></pre>

Dead commands

Following commands do nothing (FM-9 ... 13):

<pre><Command !WindowOpen** Not supported on all platforms ** <Label Open> <KeySequence \!wo></pre>
<pre><Command ViewPublisherBoundaries <Label Publisher Boundaries> <KeySequence \!v1> (lowercase L)</pre>
<pre><Command ReportCmds_byShortcut <Label Report Commands by Shortcuts> <KeySequence \!SCR2></pre>

Viewer popup command

The commands for the viewer also work in the ordinary edit window:

Command	KeySequence	Definiiton
GotoNextPage	\!pn	\x34D
GotoPreviousPage	\!pp	\x34C
GotoFirstPage	\!pf	\x340
GotoLastPage	\!pl	\x341
GotoPreviousScreen	\!vsp	\xD40
GotoNextScreen	\!vsn	\xD41

Command names from plugins and scripts

Available commands have various sources. Even native FrameMaker comes with plugins: `mapper.dll`, `masterpages.dll` etc. Most installations add scripts, both started automatically or on demand.

To be able to set up custom menus and tool bars it is necessary to get the command names from these sources. The best method is to use the free [FrameScript Report FM-commands](#) from itl.

Plugins

For example, the plugin `AutoText.dll` from Silicon Prairie Software creates commands which depend on a table in the related document `AutoText.fm`. This is an excerpt of these commands from my personal settings in the table:

Command name	Command label	Shortcut	FCode
Editors note in frame [ctrl+3]	Editors note in frame [ctrl+3]	Ctrl+3	1400DF0
Callout figure [ctrl+4]	Callout figure [ctrl+4]	Ctrl+4	1410DF0
Frame outside column [ctrl+4]	Frame outside column [ctrl+4]	Ctrl+5	1420DF0
Icon frame [ctrl+6]	Icon frame [ctrl+6]	Ctrl+6	1430DF0

ExtendScript

Note: *This example is part of the download of [FM-customisation-example.zip](#).*

ExtendScript `simple.jsx` contains the following statements and thus creates the commands listed below.

```

6 #target framemaker
7
8 var msg1 = "JavaScript alert message\nYou always get an :
9 var msg2 = "FrameMaker Alert dialogue\nIcon/Buttons depe
10
11 // set up menu with just two items
12 var mMenu = app.GetNamedMenu("!MakerMainMenu");
13 var simpleMenu = mMenu.DefineAndAddMenu("Simple", "Alert:
14 simpleMenu.DefineAndAddCommand(1,"msgJS","JS alert","");
15 simpleMenu.DefineAndAddCommand(2,"msgFM","FM Alert","");
16 UpdateMenus();
17
18 // watch the subtle difference in syntax: alert | Alert
19 function Command(cmd){
20     switch(cmd) {
21         case 1:
22             alert (msg1, "Message title");
23             break;
24         case 2:
25             Alert(msg2, Constants.FF_ALERT_CONTINUE_NOTE);
26             break;
27     }
28 }

```

Command name	Command label	FCode
msgJS	JS alert	18B0DF0
msgFM	FM Alert	18C0DF0

You see that there is no relation between the script file name and the command names.

Keyboard shortcuts for ExtendScripts

Define shortcuts

The above example does not define keyboard shortcuts for the menu entries, because at that time I did not know how to do it.

To define a shortcut the last parameter in DefineAndAddComand gets the key-sequence. Be aware to double the \ for the ESC notation (ESC,q,j and ESC,q,f). See [KeySequence](#) on page 31.

You also want to indicate these shortcuts in the menu entries. There are two methods:

Shortcut label by TAB

As known from the example ([ExtendScript](#) on page 38) the following actually precedes the Define Shortcut section in the script:

```
var oMenus = {}, menuLocation, simpleMenu;
oMenus.MenuMain = "Test Alerts";
oMenus.MenuJS = "Alert by JS\tESC,q,j";
oMenus.MenuFM = "Alert by FM\tESC,q,f";
menuLocation = app.GetNamedMenu("!MakerMainMenu");
simpleMenu = menuLocation.DefineAndAddMenu("Simple", oMenus.MenuMain);
simpleMenu.DefineAndAddCommand(1,"msgJS","JS alert","\!qj");
simpleMenu.DefineAndAddCommand(2,"msgFM","FM Alert","\!qf");
UpdateMenus();

function Command(cmd){
  switch(cmd) {
    case 1:
      alert (msg1, "Message title");
      break;
    case 2:
      Alert(msg2, Constants.FF_ALERT_CONTINUE_NOTE);
      break;
  }
}
```

Explicit definition of Shortcut label

The command property KeyboardShortcutLabel is used for this method. To me it is not clear why this is a property of the command and not of the menu item.

```
var oMenus = {}, menuLocation, simpleMenu, commandJS, commandFM;
oMenus.MenuMain = "Test Alerts";
oMenus.MenuJS = "Alert by JS";
oMenus.MenuFM = "Alert by FM";
commandJS = 1,"msgJS","JS alert","\!qj");
commandJS.KeyboardShortcutLabel = "ESC,q,j";
commandFM = 2,"msgFM","FM Alert","\!qf";
commandFM.KeyboardShortcutLabel = "ESC,q,f";
menuLocation = app.GetNamedMenu("!MakerMainMenu");
simpleMenu = menuLocation.DefineAndAddMenu("Simple", oMenus.MenuMain);
simpleMenu.DefineAndAddCommand(commandJS);
simpleMenu.DefineAndAddCommand(commandFM);
UpdateMenus();

function Command(cmd){
  switch(cmd) {
    case 1:
      alert (msg1, "Message title");
      break;
    case 2:
      Alert(msg2, Constants.FF_ALERT_CONTINUE_NOTE);
      break;
  }
}
```

2016-12-03

E:_DDDprojects\FM-toolbar00\AllIETB\etb-customising-fm12.fm

LD+D D

FrameScript

Note: This example is part of the download of [FM-customisation-example.zip](#).

Ordinary scripts get generic names (ESLRUNxxx) which may change from FrameMaker start to start. Hence a menu or toolbar using these can not be distributed to other installations.

Command name	Command label	FCode
ESLSSRUN306	Primitive	18E0DF0
ESLSSRUN307	Something Else	18F0DF0

To get a static command name to be used in a menu or toolbar the script must be set up as an EventScript.

- EventScripts must be installed, not just run.
- In an EventScript commands must reside within Events.
- Menus and commands are defined within the Event Initialize.
- It is good practice to remove the menu with Event Terminate.
- A menu item is not necessary, if the command is used in a toolbar only.

Installing FrameScript primitive.fs1 containing the following statements creates the command MyCommand1.

```

13 // A menu item is not necessary, for example if the comm
14
15 // --- Set up a command name and the event trigger
16 Event Initialize
17 // Get Object Type(Menu) Name('!MakerMainMenu') NewVar(gv
18 // New Menu Label('My Menu') NewVar(gvMyMenu) AddTo(gvMa
19 New Command Name('MyCommand1') Label('My Command 1')
20 EventProc(evtCmdEvent) NewVar(gvMyCmd1) // AddTo(gvMyM
21 EnabledWhen(EnableAlwaysEnable)
22 EndEvent
23
24 // --- Remove command and menu at de-installtion time
25 Event Terminate
26 // Remove CommandObject(gvMyCmd1) From(gvMyMenu);
27 // Remove MenuObject(gvMyMenu) From(gvMainMenu);
28 EndEvent // Terminate
29
30 // --- The event procedure is the real task
31 Event evtCmdEvent
32 If ActiveDoc = 0
33   MsgBox 'No active document --- Nevertheless: Welcome to
34 Else
35   MsgBox 'We have a document open --- Eventhough: Welcome
36 EndIf
37 EndEvent // evtCmdEvent

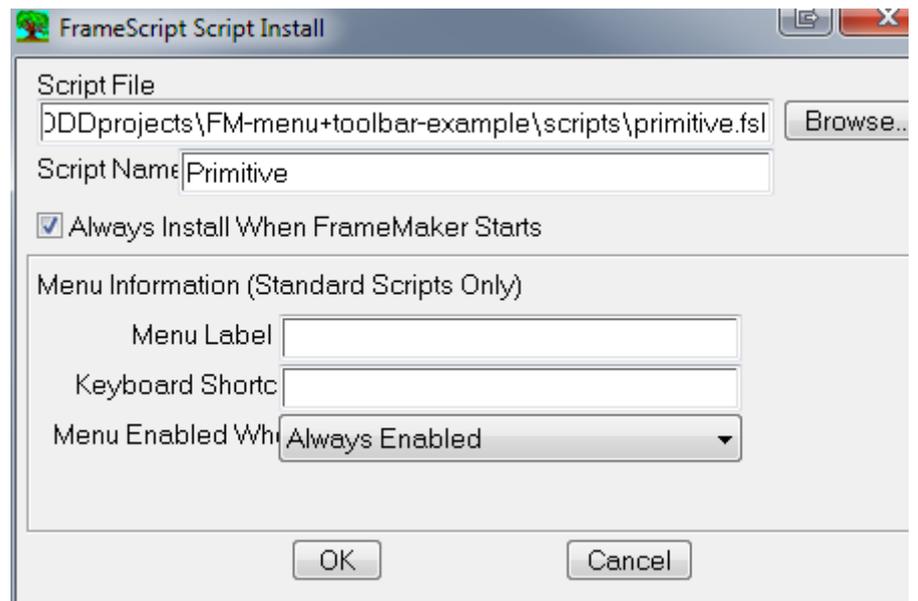
```

The installed script gets the following properties:

Command name	Command label	FCode
MyCommand1	My Command 1	A0DF0

Note: Be aware that the FCodes generated for the script may differ from FM session to FM session! Hence do not use them.

Installing the example script



Since this example script has no menu entry, you do not see an action of it as long as you do not have a toolbar with a button ...

Menu statements

Note: *Commands do not need a corresponding menu entry. They may be invoked by a keyboard short cut or a button only.*

Menu

A new menu or sub menu is defined with this statement:

Syntax `<Menu menu-name <Label menu-label>>`
 Examples `<Menu ETBmenu <Label Enhanced Toolbar (ETB)>>`
`<Menu ETBlocalDocuments <Label Local documentation (pdf)>>`

Reserved menu

Many of the menus defined in the standard menu files for FrameMaker are reserved menus. FrameMaker has intrinsic knowledge about reserved menus; it can refer to these menus directly by name.

Syntax `<ReservedMenu !menu-name <Label menu-label>>`
Menu-name FrameMaker relies on Permanent menus and also on Context menus. Both are defined with the ReservedMenu statement in the standard menu files. Custom menus must not use names of these reserved menus. It is a good idea to prefix custom menu names by a project abbreviation. For example, **ETBmenu** or **!_MTmenu**.
Menu-label The Label detail is the same as for commands, because the command is represented in the menu item. See [Label](#) on page 30.
Reserved menus By convention, the names of reserved menus begin with an exclamation point (!). The highlighted items in the table are permanent menus²².

Menu ID	Description
!BookMainMenu	Menu bar for complete menus (book window active)
!CustomMakerMainMenu	Menu bar for custom menus (document window active)
!EmbeddedObjectMenu	Submenu Edit > Object
!HelpMenu	Menu Help
!MakerMainMenu	Menu bar for complete menus (document window active)
!MenusMenu	Submenu View > Menus
!MSWindowMenu	Menu Window
!QuickBookMainMenu	Menu bar for quick menus (book window active)
!QuickMakerMainMenu	Menu bar for quick menus (document window active)
!RulerAlignMenu	¶-alignment pop-up menu in the formatting bar
!RulerAlignMenu	¶-alignment pop-up menu in the formatting bar
!RulerControlMenu	Formatting bar: commands !ShowRulerToggle, !ShowRulerAlignmentSpacingAndTabs and !ShowRulerParagraphTags - all with an empty definition (\x0).
!RulerParaMenu	¶-formats pop-up menu in the formatting tool bar
!RulerSpaceMenu	Line-spacing pop-up menu in the formatting tool bar
!StructureViewMainMenu ^a	Structure menu bar (structured product interface only)
!ViewOnlyMainMenu ^a	View-only menu bar

a. Not used in any of the FM-13 *cfg files - depreciated?

²² This table is deducted from the various cmdxxx.cfg files of FM-13.

Permanent menus

FrameMaker relies on some menus existing. In the table of Reserved Menu their ID is bold. You cannot remove these menus from a menu configuration file. FrameMaker will not work properly without them.

Context menus

The default context menu for a particular selection does not contain every possible command. If a menu item is not applicable to the selection and the current state of FM, the menu item will be dimmed.

Context menus can not contain shifted commands. See [ShiftCommand](#) on page 28).

Context menus can also be displayed by pressing **Shift+F10**. In this case, the appearing menu depends on whether there exists a selected object, an insertion point, or neither.²³⁾

Context menu id	Object providing the context
!AnchoredFrameContextMenu	an anchored frame
!AutoSpellCheckContextMenu	
!BookContextMenu	a book window
!DocumentContextMenu	the document as a whole (no active insertion point and nothing selected)
!EmbeddedObjectContextMenu	an OLE object
!GraphicsContextMenu	all graphic objects except an anchored frame
!MathContextMenu	an equation
!MultiGraphicsContextMenu	any grouped graphic object
!QuickBookContextMenu	a book window when Quick Menu is the current menu set up
!StructureContextMenu	the Structure View in FrameMaker+SGML
!StructuredTextContextMenu	text in a structured text flow in FrameMaker+SGML
!TableContextMenu	a table
!TableTextContextMenu	text in a table
!TextContextMenu	text
!TextLineContextMenu	text line selected as text
!TextLineGraphicContextMenu	text lines selected as graphic objects
!ViewerPopup	a View-only document window
!ViewOnlyBookContextMenu	a View-only book window
!ViewOnlyDitamapContextMenu	DitaMap
!ViewOnlyDitamapContextMenuContainer	
TableMenuInTextContextMenu	

Add

This statement adds an item to a menu at the end of the already existing entries. Hence normally a corresponding Order statement exists in a customisation file.

Syntax

```
<Add command <Menu menu-name>>
<Add submenu-name <Menu menu-name>>
<Add Separator <Menu menu-name>>
```

Command This identifies an already defined command.

Submenu-name This identifies an already defined menu.

²³ This table is deducted from the various cmdxxx.cfg files of FM-13. See also *FP_EnabledWhen value* in the FDK reference.

<i>Menu-name</i>	Name of the menu to which the command or sub-menu shall be added.
<i>Separator n</i>	Within a menu the separators must be numbered to get unique names for an <i>Order</i> statement.
<i>Examples</i>	<pre><Add Open <Menu FileMenu>> <Add ETBspecial <Menu ETBmenu>> <Add Seaprator5 <Menu ETBmenu>></pre>

Order

This statement defines where a menu entry shall be placed in the menu.

<i>Syntax</i>	<pre><Order menu.new-item <First menu>> <Order menu.new-item <Last menu>> <Order menu.new-item <Before menu.ref-item>> <Order menu.new-item <After menu.ref-item>></pre>
<i>Menu</i>	The menu we are talking about, defined by a <i>Menu</i> statement.
<i>New-item</i>	The item to be placed. It has been defined in the <i>menu</i> .
<i>Ref-item</i>	The reference item in <i>menu</i> relative to which the new item is inserted.
<i>Examples</i>	<pre><Order !TextLineContextMenu.SpecialCharsContext <Before !TextLineContextMenu.Undo>> <Order !TextLineContextMenu.Separator1 <Before !TextLineContextMenu.Undo>> <Order ViewMenu.ViewGraphicsOn <Before ViewMenu.Separator3>></pre>

Workspace definition

Workspaces are defined in files `xxx.cfws` (current) and `xxx.fws` (last saved). In most cases we need not create or modify (besides line 2) such a file.

Modify or create workspace

Menus and toolbars appear in a view due to the file location of them. Multiple workspaces can be defined for a particular view. Activating a newly introduced toolbar (docked or undocked) modifies the current workspace, which can be saved to a new name.

workspace file

The following is the definition found in `Custom.cfws` of the example download (long lines are truncated to display the structure):

```

1 <FrameUI version="1">
2   <data type="all" menuFile="custom_menus.cfg" toolbarFile="custom_toolset.xml"/>
3   <fm-workspace>
4     <workspace version="1">
5       <dock anchor="left" content="palette toolbar" is-closed="false"/>
6       <dock anchor="right" content="palette toolbar" is-closed="false">
7         <tab-pane mode="expanded" preferred-iconic-length="0" layout-mode="auto-flow">
8           <tab-group active-palette="00070CB2" is-closed="false">
9             <palette id="00070CB2" is-closed="false" preferred-unconstrained-size="135
10            <palette id="00060F46" is-closed="false" preferred-unconstrained-size="135
11            <palette id="000A10CE" is-closed="false" preferred-unconstrained-size="135
12            <palette id="00060BAA" is-closed="false" preferred-unconstrained-size="135
13          </tab-group>
14          <tab-group active-palette="00060DD6" is-closed="false">
15            <palette id="000C1118" is-closed="false" preferred-unconstrained-size="317
16            <palette id="00060DD6" is-closed="false" preferred-unconstrained-size="359
17          </tab-group>
18        </tab-pane>
19      </dock>
20      <dock anchor="top" content="multi-control-bar" is-closed="false">
21        <control-bar-pane >
22          <control-bar id="00060B9A" origin="4 35" size="858 34" is-closed="false" app-
23        </control-bar-pane>
24      </dock>
25      <dock anchor="bottom" content="palette" is-closed="false"/>
26    </workspace>
27  </fm-workspace>
28 </FrameUI>

```

Key in the workspace file are the `palette` and `control-bar` statements, which hold various information about the panels and toolbars.

Note: For customisation we are normally only dealing with the second line which defines the menu set and the toolbar set. Be aware that xml comments are removed from workspace files!

Size of toolbar

Only if a drop down box appears in the toolbar you may need to adjust the size parameters in the `control-bar` statement. You do this after the workspace file has been created. Normally you adjust the parameters `size = preferred-size` and then minimum and maximum.

```
<control-bar id="004C0966" origin="874 31" size="255 34" is-closed="false" app-
```

```
data="#lt;control-bar cb-data=#quot;custom_tb#quot; minimum-size=#quot;240 34#quot; maximum-size=#quot;300 34#quot; preferred-size=#quot;255 34#quot; /> />
```

Custom menu

If you want to refer to a custom menu you will change the references in line 2, for example

from

```
<data type="all" menuFile="menus.cfg" toolbarFile="fmttoolbar.xml"/>
```

to

```
<data type="all" menuFile="custom_menus.cfg" toolbarFile="fmttoolbar.xml"/>
```

You also need to set up a menu file `custom-menus.cfg`. and place it in appropriate directory.

Custom toolbar set

If you want to refer to a custom toolbar set, you will change the references in line 2, for example

from

```
<data type="all" menuFile="menus.cfg" toolbarFile="fmttoolbar.xml"/>
```

to

```
<data type="all" menuFile="menus.cfg" toolbarFile="custom_toolbar.xml"/>
```

You will set up a corresponding list of toolbars in `custom_toolset.xml`. This list refers to the individual tool bar files.

Note: *Toolbars containing a drop-down list can not be docked left or right to become oriented vertically!*

Initialisation files maker.ini and others

The *.ini files are a standard Windows initialisation files which are divided into sections. Section names are enclosed in brackets. The equal sign may be surrounded by spaces:

```
[Frame]
ProductInterface=FrameMaker
; Comment lines start with a semicolon.
```

The coding of these files vary. Most use the Windows cp 1252, maker.ini uses UTF-8.

When opening an initialisation file in FrameMaker, assure to open it as **Text**. When you're finished editing the file, use the **Save As** command to save it as **Text Only**.

For FM-13 the maker.ini file comprises about 350 entries. Some of them are necessary to get back the behaviour of previous FM-versions , for example

```
SymbolSortingBeforeAlphaNumeric = On
```

Since FM-7.1 the file in \$HOME is considered the master file and only the file in the user-area is modified individually. These files are now coded in UTF-8. Nevertheless special characters can be defined in the traditional way using :

```
FindSpaceBefore=On !%),. : ; ? ] } \u00bb\u201d\u2019\u203a
;SmartQuotes \xd4\xd5\xd2\xd3 ) English curved quotes
SmartQuotes='’"”
```

The last line shown above uses the Unicode of the characters, while the comment line uses the traditional FrameRoman notation.

Of course it is possible to establish a complete copy of the \$HOME file in the user-area and then open FrameMaker. However, some entries, such as ProductInterface are not present in the main file. They describe items which might differ between users of the same system.

Other ini files

Special features are supported by additional ini files²⁴:

Initialisation file	Purpose, used by
AuthorView.ini	Set-up of the Author View
ditafm-output.ini	Set-up for DITA output processing
ditafm.ini	Set-up for DITA processing
DynaHelpPreview.ini	Publisher: Dynamic Help extension
MathFlowPlugin.ini	Set-up of the none-Adobe plug-in MathFlow
ModalDialogPosition.ini	Save the current positions of the modal dialogues
pdfsize.ini	Settings from Format > Document > Optimise PDF Size > Options
SkinGallery.ini	Publisher: skins (WebHelp, WildFire, AirHelp)
sqPalmXmlViewer.ini	Publisher: Palm xml viewer
sqSkin.ini	Publisher: sqSkin extension
WHATSTHS.INI	Set-up for the What's This function of Publisher
XmlCodeView.ini	Set-up of the XML Code View

²⁴ List according to FM-13 installation.

Entries in ini files

To avoid redundancy I do not list the entries of `maker.ini` and other `ini` files here. Look at www.daube.ch for a maintained list.

Note: *Since FM-12 Adobe maintains a document describing the entries in `maker.ini` - See **Help > Help Topics > 5th icon (FrameMaker Help Center) > FrameMaker Resources.***

2016-12-03

E:_DDDprojects\FM-toolbar00\AllIETB\etb-customising-fm12.fm

DDD

