

Modelit  
Elisabethdreef 5  
4101 KN Culemborg  
+31(345)531717



info@modelit.nl  
www.modelit.nl

# **Modelit Layout Manager for Matlab**

Version: 2008\_01  
Date: August 13, 2008

# Model!T

Version: 2008\_01  
Date: August 13, 2008  
Manual: Modelit Layout Manager for Matlab  
Author: Nanne van der Zijpp, Kees-Jan Hoogland  
Copyright: 2008, Modelit  
Contact: [info@modelit.nl](mailto:info@modelit.nl)  
[www.modelit.nl](http://www.modelit.nl)

1	Introduction.....	1
1.1	Introducing the Modelit Layout Manager for Matlab.....	1
1.1.1	Gui's en Guide .....	1
1.1.2	Complex GUI's: Layout Manager .....	1
1.1.3	Layout Manager features.....	1
1.2	Comparison with the Matlab uipanel object.....	2
1.3	Help.....	2
1.4	System requirements.....	3
1.5	How to proceed from here.....	4
2	Installation.....	5
3	Getting started with the Layout Manager.....	6
3.1	Step 1: Positioning frames.....	6
3.1.1	A GUI example.....	6
3.1.2	Creating the GUI example.....	8
3.1.3	Specifying frame attributes: <code>lm_createframe</code> .....	9
3.2	Step 2: Positioning objects in a frame.....	10
3.2.1	Specifying object attributes: <code>lm_linkobj</code> and <code>lm_arrange</code> .....	11
3.3	Summary.....	12
4	Layout Manager Reference Manual.....	13
4.1	<code>lm_arrange</code> .....	13
4.2	<code>lm_childframes</code> .....	17
4.3	<code>lm_createframe</code> .....	17
4.4	<code>lm_deleteframe</code> .....	19
4.5	<code>lm_deleteframecontent</code> .....	19
4.6	<code>lm_divider</code> .....	19
4.7	<code>lm_doubleframe</code> .....	20
4.8	<code>lm_exitbutton</code> .....	21
4.9	<code>lm_enableonoff</code> .....	22
4.10	<code>lm_exittext</code> .....	22
4.11	<code>lm_framelist</code> .....	23
4.12	<code>lm_frameonoff</code> .....	23
4.13	<code>lm_initaxes</code> .....	24
4.14	<code>lm_innerpixelsize</code> .....	24
4.15	<code>lm_isparent</code> .....	24
4.16	<code>lm_lineprops</code> .....	24
4.17	<code>lm_linkobj</code> .....	25
4.18	<code>lm_linkslider2frame</code> .....	28
4.19	<code>lm_listframeHandles</code> .....	28
4.20	<code>lm_parentframe</code> .....	29
4.21	<code>lm_patchprops</code> .....	29
4.22	<code>lm_pixelsize</code> .....	29
4.23	<code>lm_resize</code> .....	29
4.24	<code>lm_shifrank</code> .....	29
4.25	<code>lm_sortframes</code> .....	30
4.26	<code>lm_title</code> .....	31



# 1 Introduction

## 1.1 Introducing the Modelit Layout Manager for Matlab

The Modelit Layout Manager for Matlab (in short: Layout Manager) is a tool to position Matlab handle graphic objects such as uicontrols and axes within a figure. It has been designed for Matlab developers who want to provide their Matlab applications with intuitive and powerful user-interfaces.

The objectives of the toolbox are:

- to build interfaces that look good on a variety of displays and allow figure resizing;
- to build interfaces in an efficient way;
- to build interfaces in a modular way;
- to support extra features like tabs and sliders.

All objects positions are specified relative to frame objects, using both normalized and pixel coordinates. Frame objects are nested in hierarchical structure, where each frame may have other frames as their children. When a figure is created, modified or resized, all frame positions and the object positions are recomputed in a split second and the new positions are shown instantaneously.

### 1.1.1 Gui's en Guide

Matlab comes with a useful Wysiwyg editor for designing user interfaces (GUIDE). However, using GUIDE for building complex or real-time configurable interfaces is not recommended, because the time needed to build an interface and align all objects increases exponentially with the number of elements included in the interface. This imposes a natural limit to the complexity that can be achieved. Moreover, it is difficult to build GUIDE interfaces that can be resized, parameterized, or re-used.

### 1.1.2 Complex GUI's: Layout Manager

The Modelit Layout Manager takes care of locating and aligning objects in so called "frames". A GUI is made up of an arbitrary set of frames that are hierarchically organized. Once defined, frames may be exchanged between applications. This makes it relatively easy to re-use parts of interfaces in new applications.

### 1.1.3 Layout Manager features

The Layout Manager supports the following features:

- Link Matlab Handle Graphic objects to a frame;
- Absolute, normalized or mixed position definition;
- Aligning objects in a grid;
- Automatic setting of minimal frame size;
- Nesting frames;
- Tabs and sliders;
- Properties "visible" and "enabled" supported at frame level;
- Automatic resizing of GUI's;
- Adjustment of GUI to different window size;
- Inspection of properties and easy access to matlab code (see Figure 1).

The Layout Manager does *not* do any of the following:

- Change properties of Handle Graphic objects, other than position and enabled status;

- Change contents of axes objects;
- Implement new GUI elements like trees and tables. You need the "Modelit User Interface Components Toolbox for Matlab" for this purpose.

## 1.2 Comparison with the Matlab uipanel object

There are a number of parallels between the frame object in the Modelit Layout Manager and the Matlab uipanel object. Matlab programmers already familiar with uipanel will find it easy to get acquainted with the usage of frames. Comparing the properties of Matlab uipanel object and the Layout Manager frame object results in the table below:

Feature	uipanel object	frame object
position handle-graphic objects relative to parent using units = normalized	yes	yes
position HG objects relative to parent using units = pixels	yes	yes
position HG objects relative to parent using units = normalized+pixels	no	yes
position objects that normally appear in axes, like line and text objects, in an invisible overlaid axes	no	yes
position objects relative to parent using any of the following units: inches, centimeters, points or characters	yes	no
built in support for clipping	no	yes
built in support for sliders	no	yes
distribute objects over grid	no	yes
compute frame size from child objects	no	yes
compute frame size from child frames	no	yes
hide frame including children and child frames	no	yes
disable or enable frame including children	no	yes
position frame with position vector	yes	no
position frame using grid layout	no	yes

## 1.3 Help

### *PDF documentation*

The current document provides the user guide and reference manual and is provided as a PDF document.

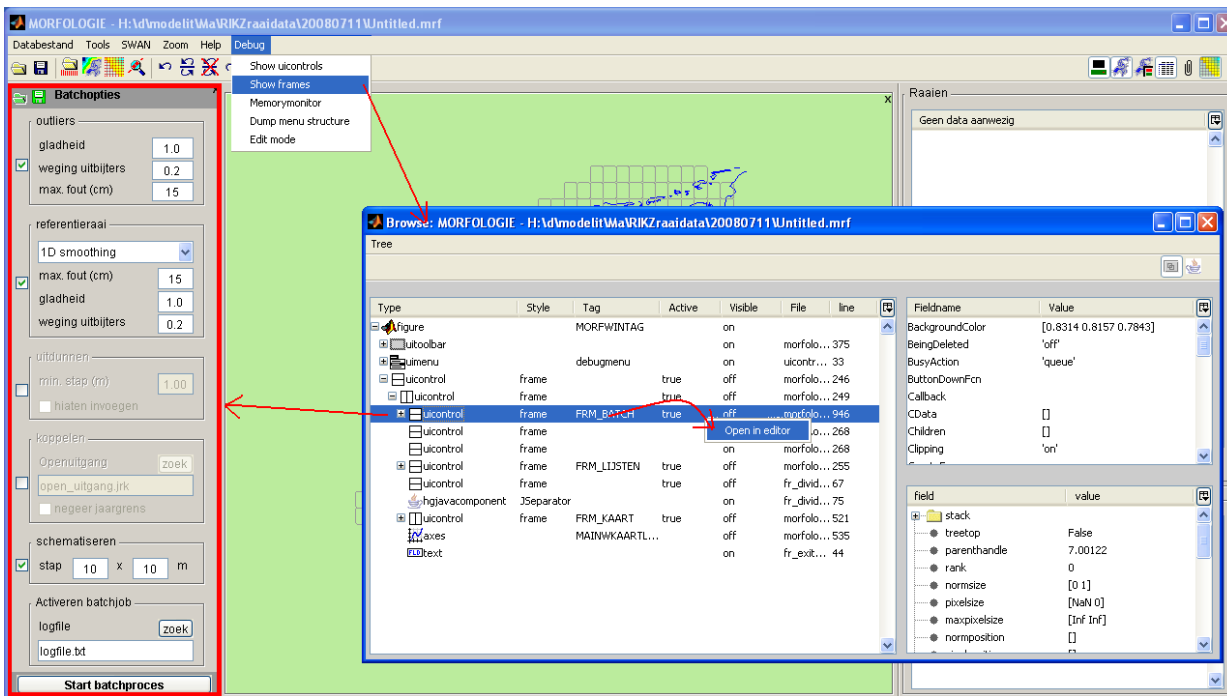
### *Manual pages*

Each function that is a part of the Layout Manager has an extensive manual page that is called by typing "help function\_name".

### *Interactive*

The Layout Manager comes with an interactive tool that analyses the frames and objects that are defined in a figure and presents these in a tree as shown in Figure 1. This tool is particularly useful for analysing more complex GUI's.

*Note: This tool is implemented in Matlab and uses components of the "Modelit User Interface Components Toolbox for Matlab". If this toolbox is not installed on your system, the tool will not work.*



**Figure 1:** Frame analysis tool. The tool highlights frames and objects and allows opening the source code at the point where each frame is defined.

## 1.4 System requirements

The Modelit Layout Manager is implemented in Matlab, and has been tested with all Matlab versions ranging from 2006a to 2008a. No undocumented Matlab features have been used, hence the software should also work with future Matlab versions without any modifications.

Any application based on the Layout Manager can be compiled to standalone applications with the Matlab Compiler.

The Layout Manager works seamlessly with other toolboxes and subroutine libraries provided by Modelit. Two toolboxes are particularly useful when building user interfaces:

- *Modelit User Interface Components Toolbox for Matlab.* This toolbox extends the standard available Matlab interface elements with tabbed panes, sortable tables, trees and so forth without the need to switch to a different programming language or complex code;
- *Modelit Application Framework for Matlab.* This toolbox implements among others an automated link between datamodel and visualization and undo/redo functionality.

Users who combine the Layout Manager with other Modelit products experience extra synergies. For example: A separator is part of the UIC-toolbox and is a visual divider between two areas in a figure. When in addition to the UIC-toolbox the Layout Manager is available, the divider can be made "draggable" just by adding one line of code. This allows users to change the layout of their interface interactively.

## 1.5 How to proceed from here

Depending on your needs you may read the following chapters:

- Chapter 2 contains all information for installing the toolbox;
- Chapter 3 is a quick introduction to using the toolbox. After having read this chapter programmers should be able to build their first application using the toolbox;
- Chapter 4 contains a full reference manual;



## 2 Installation

Installing the Modelit Layout Manager for Matlab consists of copying the directories included in the m-file distribution to target directories on your system and including these directories in your matlab path. For convenience a file `install.m` is included in the toolbox that sets the path.

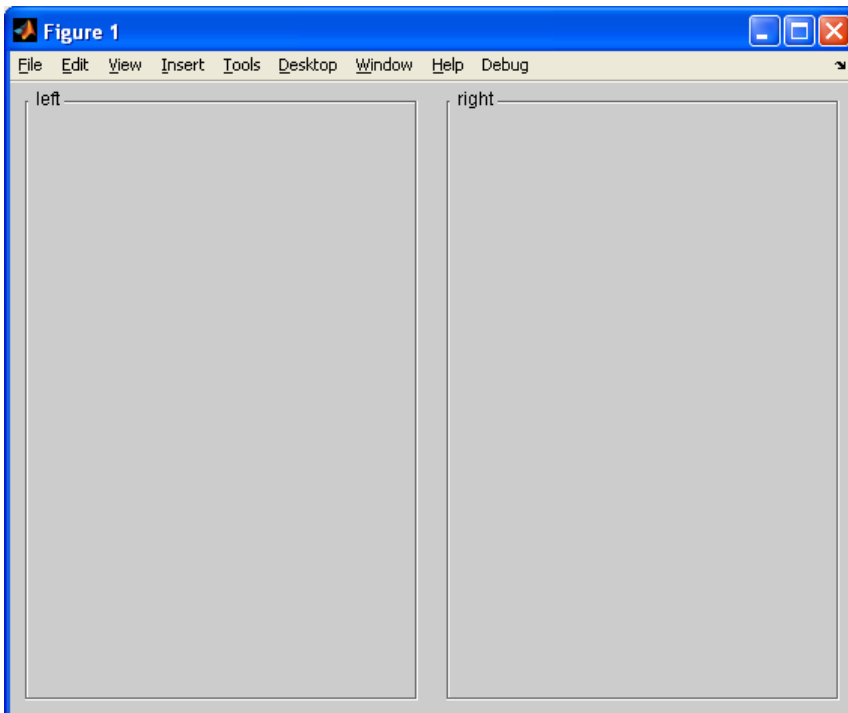
The next steps will install the Modelit Application Framework:

1. Unzip the files from the MLM.zip file.  
This creates a folder 'modelit' with subdirectories.
2. Find and run `install.m`. This prints matlab code that will include the required directories in your Matlab path on the console.
3. Copy these lines to the `startup.m` file
4. Run `startup.m` or restart Matlab

You may verify the installation by typing:

```
h=lm_createframe('figure');
lm_createframe(h,'title','left','rank',1,...
'lineprops',lm_lineprops,'minmarges',[10 10 10 10]);
lm_createframe(h,'title','right','rank',2,...
'lineprops',lm_lineprops,'minmarges',[10 10 10 10]);
lm_resize;
```

A figure with 2 visible frames should appear (see Figure 2).



**Figure 2:** *Example*

### 3 Getting started with the Layout Manager

The Layout Manager controls the appearance of an interface in a 2 stage process:

- It determines the position of a number of user defined frames;
- It determines the position of interface components relative to the frame positions and displays them.

#### 3.1 Step 1: Positioning frames

##### 3.1.1 A GUI example

The reasons for using a Layout Manager are best illustrated with a practical example. Consider the GUI shown in Figure 3. This GUI displays a map and 9 frames with alphanumerical data:

- The figure is organized in 3 columns;
- The left column is split vertically in 6 frames;
- The middle column is not subdivided;
- The right column is split vertically in 3 frames.

The width of the left column should correspond to its contents. If it is too narrow, edit boxes and buttons will not fit. If it is too wide the column will have an odd appearance with much empty space. The width of the middle column is flexible as it does not contain any fixed size objects. The width of the right column is flexible to some extent, but it should have a minimum width to display properly. Figure 4 shows how these requirements work out if the figure is resized: column 1 and 3 keep their width and the middle column changes in size.

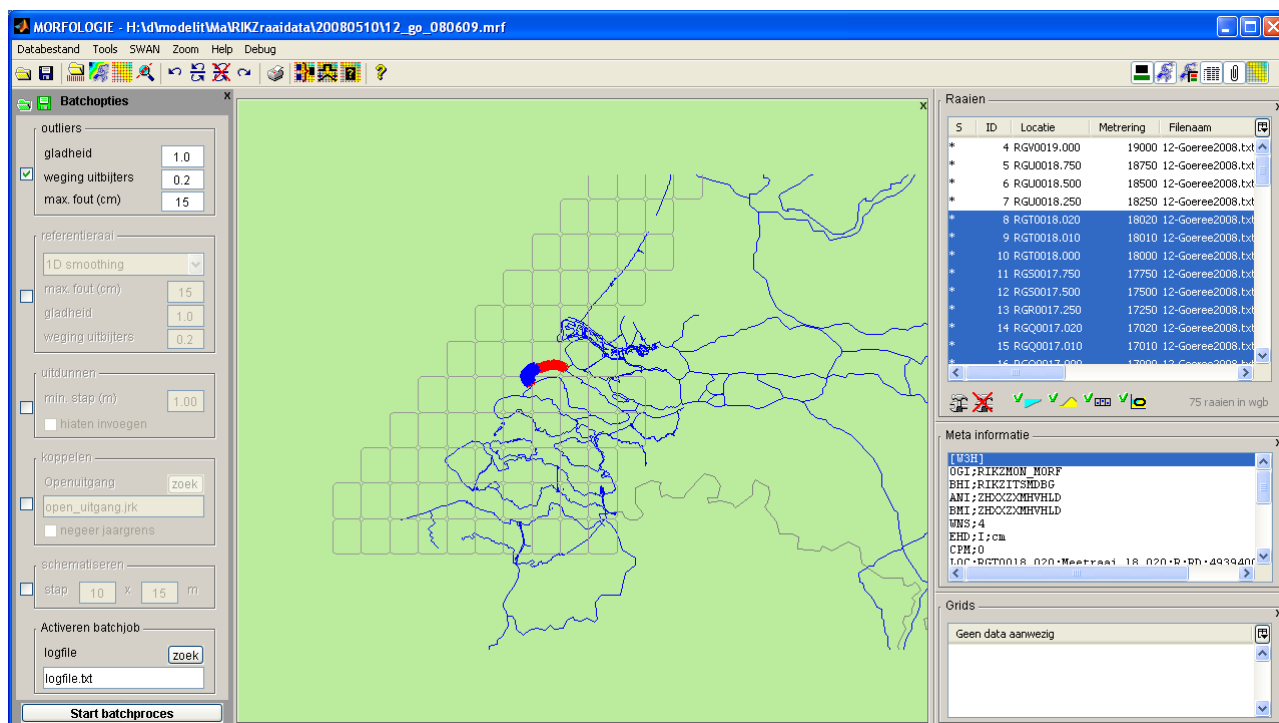


Figure 3: GUI example

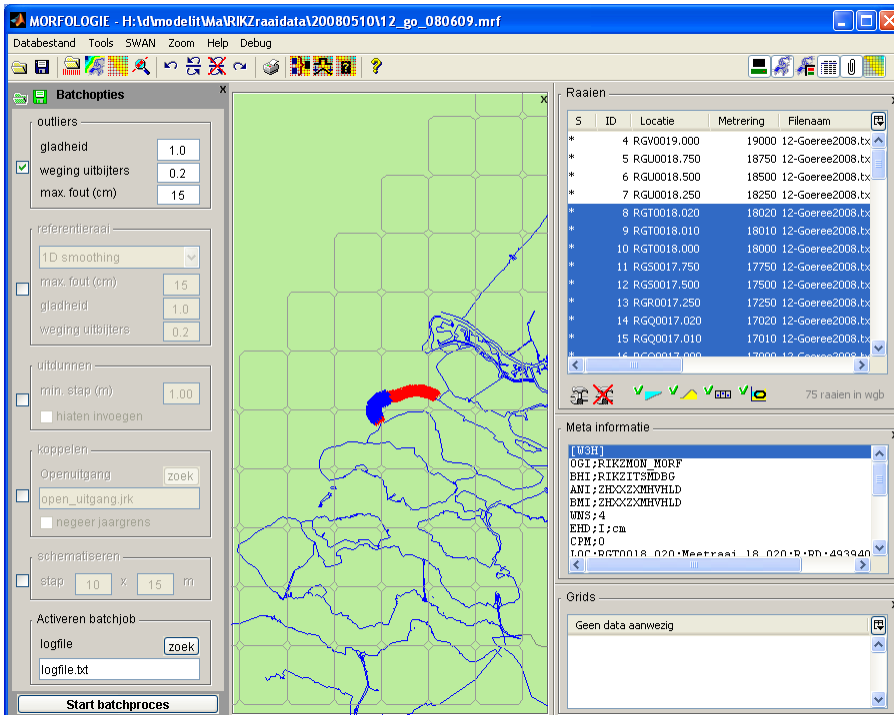


Figure 4: GUI example after resize

Not all frames displayed in Figure 3 are needed all the time and users may wish to de-activate unused frames to free up display space. In this case the expected behaviour is that the freed up space is occupied by the remaining frames in a logical manner.

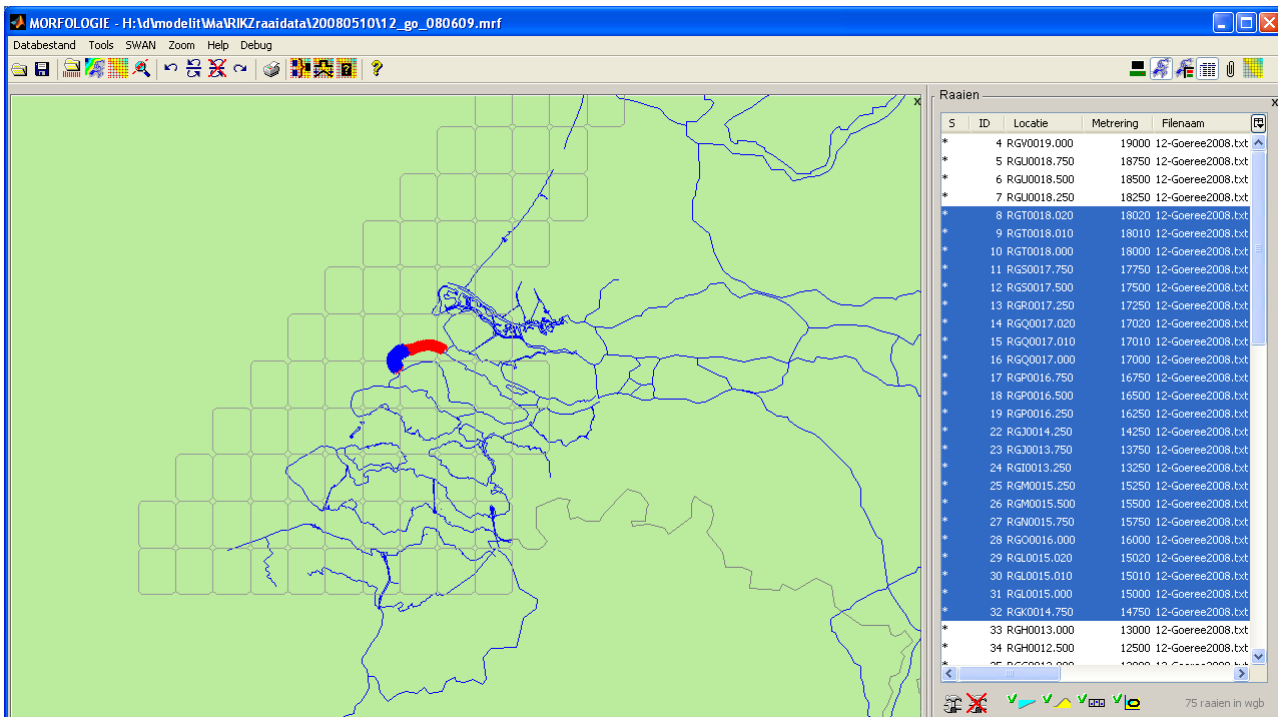


Figure 5: GUI example after de-activating unused frames

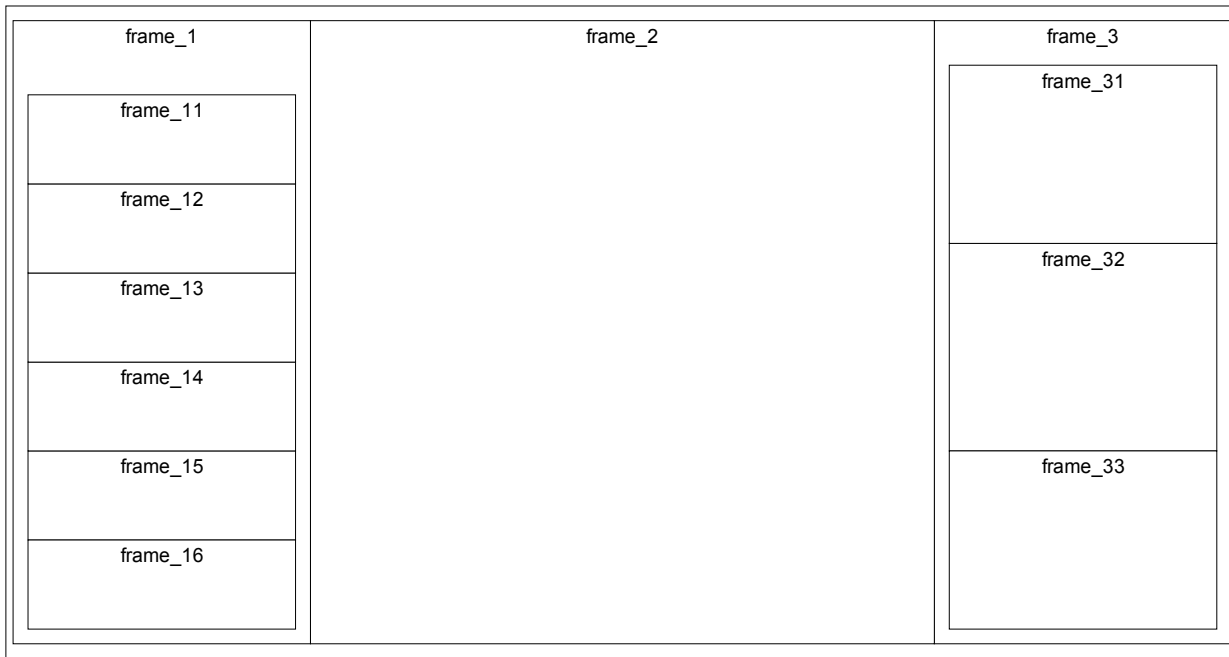
### 3.1.2 Creating the GUI example

The example shown in the previous section shows a typical GUI and its expected behaviour. The Layout Manager provides the tools for building this and similar interfaces in an easy manner.

The GUI shown in Figure 3 is schematized in Figure 6. The schema shows that the frames are organized in a nested way. Each frame has a parent and potentially one or more children. This nested structure may also be represented in a more abstract form as a tree, see Table 1. The frame sizes can be controlled with four attributes per frame:

Frame property [ <i>keyword</i> ]	What it controls
split direction [ <i>splithor</i> ]	This property determines if child frames will be organized vertically or horizontally. If this attribute is set to <i>true</i> the child frames will divide the parent frame in a horizontal direction.
rank property [ <i>rank</i> ]	This property determines the relative position of frames within a nest. If the parent frame is split in a horizontal direction, the frame with the lowest rank is displayed left. If the parent frame is split vertically the frame with lowest rank is displayed at the top position.
pixel size [ <i>pixelsize</i> ]	This is a 2 element vector referring to the minimal required width in pixels in the horizontal and vertical direction respectively. NaN may be specified to tell the Layout Manager to compute the value by adding the sizes of the child frames.
normalized size [ <i>normsize</i> ]	This is also a 2 element vector. The available space in the parent frame after deduction of the pixel sizes will be distributed proportional to the specified normalized sizes.
active property [ <i>active</i> ]	If this attribute is <i>false</i> the frame and all its child frames will not be displayed. This is regardless of the active status of the child frames.

These four attributes are sufficient to get started with the Layout Manager. Additional properties will let you control an interface in more detail, and are described in the reference manual.



**Figure 6:** Schematic GUI-layout

**Table 1:** Underlying specification of the example GUI in Figure 5

tree structure	split direction	rank	pixel size	normalized size	Active
root frame	horizontal	1	[NaN NaN]	[1 1]	yes
+----frame 1	vertical	1	[NaN NaN]	[0 1]	no
+----frame_11	horizontal	1	[200 100]	[1 0]	yes
+----frame_12		2			
+----frame_13		3			
+----frame_14		4			
+----frame_15		4			
+----frame_16		6			
+----frame_2	not applicable	2	[0 0]	[1 1]	yes
+----frame_3	vertical	3	[NaN NaN]	[0 1]	yes
+----frame_31	not applicable	1	[300 100]	[1 1]	yes
+----frame_32		2			no
+----frame_33		3			no

### 3.1.3 Specifying frame attributes: `lm_createframe`

The frame attributes mentioned in Table 1 are specified when a frame is created with the toolbox function `lm_createframe`:

```
lm_createframe(Parent,Property1,Value1, Property2,Value2,...)
```

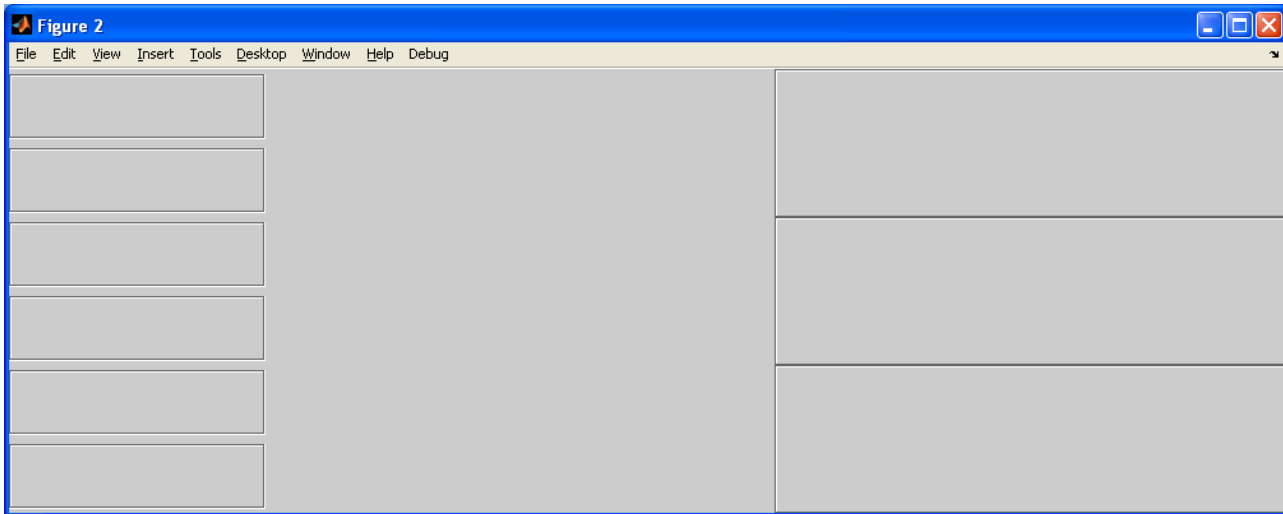
In the current example the properties "rank", "normsize", "pixelsize" and "lineprops" are used. A full description of all available parameters and their uses is available in the reference manual chapter.

The code below generates the layout shown in Figure 6.

```

HWIN=figure('pos',[20 20 1000 650],'units','pixels','ResizeFcn',@lm_resize);
hRoot=lm_createframe(HWIN);
h1=lm_createframe(hRoot,'rank',1,'pixelsize',[NaN NaN], 'normsize',[0 1])
for k=1:6
    lm_createframe(h1,'rank',k,...
        'pixelsize',[200 50],...
        'normsize',[0 0],...
        'lineprops',lm_lineprops);
end
h2=lm_createframe(hRoot,'rank',2);
h3=lm_createframe(hRoot,'rank',3);
for k=1:3
    lm_createframe(h3,'rank',k,'lineprops',lm_lineprops);
end
lm_resize(HWIN);

```

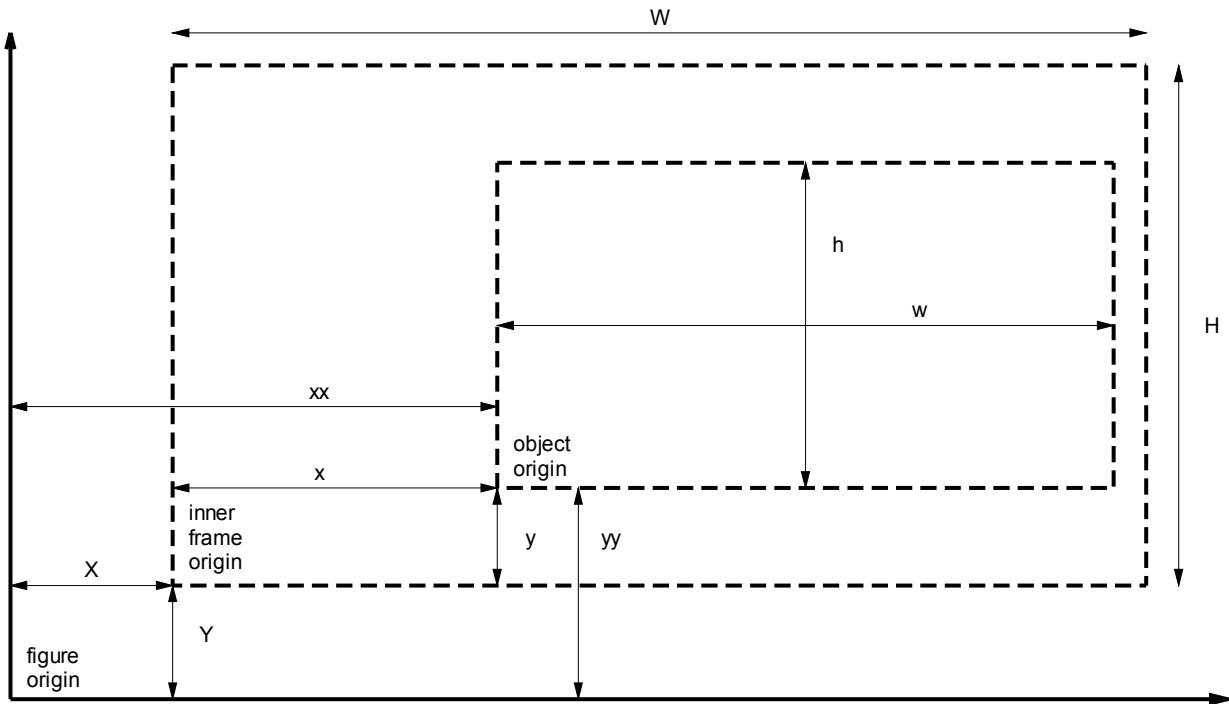


**Figure 7:** Result of the `lm_createframe` example.

### 3.2 Step 2: Positioning objects in a frame

Once the frames are in place, the positions of the objects are specified relative to the position of the frame. The mechanism that determines the exact position of an object in a figure will be explained using the example in Figure 8. This example identifies the following parameters (all in pixels):

- X: x-position of Lower-Left (LL) corner of innerframe. The innerframe equals the outerframe after margins are applied. These margins are specified or defaulted when the frame is created. For now it suffices to know that X is a result of step 1.
- Y: y-position of Lower-Left (LL) corner of innerframe.
- W: width of inner frame.
- H: height of inner frame.
- x: horizontal distance between LL-corner of frame and LL-corner of object.
- y: vertical distance between LL-corner of frame and LL-corner of object.
- w: width of object.
- h: height of object.



**Figure 8:** Parameters that determine the position of interface components within a figure.

Parameters  $X$ ,  $Y$ ,  $W$  and  $H$  follow from step 1. These parameters are a starting point for the current step that positions the interface elements.

The position of an object depends on normalized and pixel coordinates. Normalized coordinates define position and size *proportional* to the size of the frame. Pixel coordinates define *absolute* position and size in pixels. The position of an object is the sum of both, and is computed as follows.

Define:

specified normalized position:  $[xN, yN, wN, hN]$   
 specified pixel position:  $[xP, yP, wP, hP]$   
 resulting object pixelposition:  $[xx, yy, w, h]$

Then the object position is computed as follows:

$$\begin{aligned} xx &= X + xN * W + xP \\ yy &= Y + yN * H + yP \\ ww &= wN * W + wP \\ hh &= hN * H + hP \end{aligned}$$

### 3.2.1 Specifying object attributes: `lm_linkobj` and `lm_arrange`

Two methods can be used to specify the normalized and pixel position properties of an object:

- Specify coordinates explicitly. The position and size of the object are specified explicitly;
- Specify position in grid. The row and column of the object in an imaginary grid are specified. The Layout Manager takes care of specifying the coordinates.

Note that in the end both methods operate on the same set of attributes. The first method provides more control while the second method saves programming effort.

Both methods require that the command `lm_linkobj` is used to specify an object as a child of a frame.

Method	Relevant attributes	Example
Specify coordinates explicitly	normpos pixelpos	<code>lm_linkobj(h,hFrame,... normpos,[.5,.5,0,0],... 'pixelpos',[0 0 20 10]);</code>
Specify grid position	row col	<code>lm_linkobj(h,hFrame,'row',a,'col',b) lm_arrange(hFrame)</code>

### 3.3 Summary

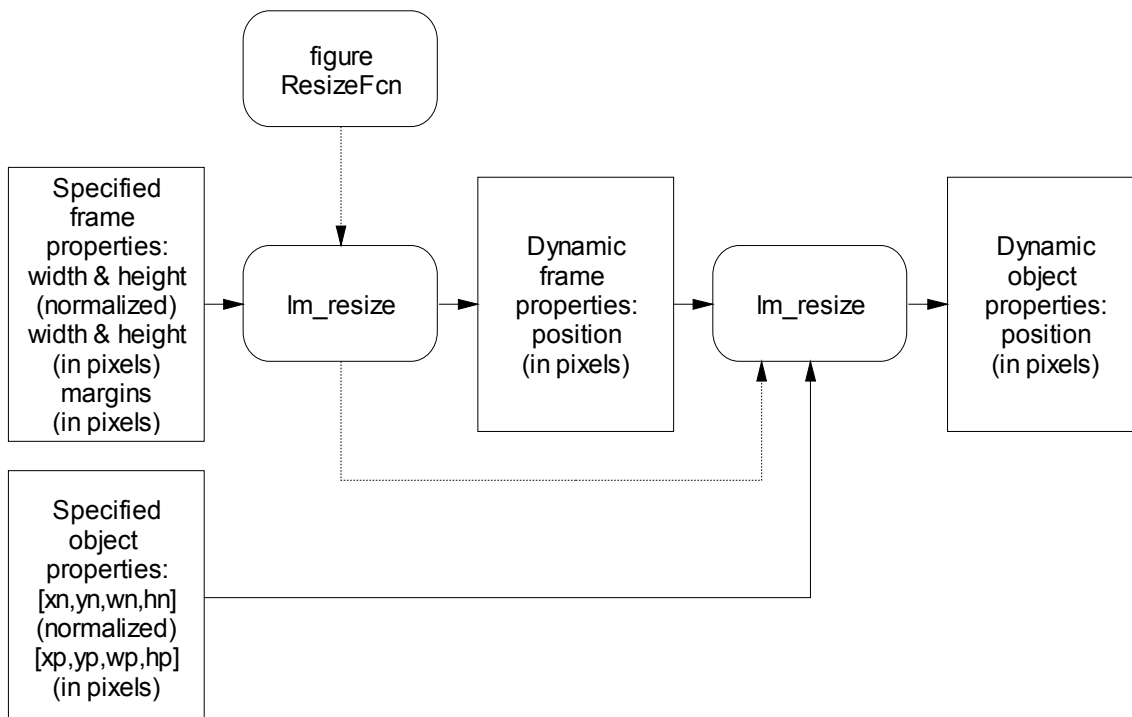
The architecture of the Layout Manager distinguishes properties of the following categories:

- *Specified properties.* These properties are user-specified and stored as attributes of objects and frames.
- *Dynamic properties.* These properties are computed by the function `lm_resize` every time a figure is resized or one or more elements are changed.

The properties of the second category coincide with the set of properties that one would use to control an interface if the Layout Manager is not available. Central to the toolbox is the function `lm_resize`. This function is responsible to position all frames and objects. This function should be called every time:

- an element is added to the GUI;
- the size of an element is changed;
- the visibility of an element is changed;
- the size of the figure is changed.

For this reason it is recommended to set the figure `ResizeFcn` to `lm_resize`. Figure 9 illustrates the process of computing the dynamic properties as described in this chapter.



**Figure 9:** Architecture of the Layout Manager Toolbox: the function `lm_resize` computes dynamic object properties based on figure size and specified properties.



## 4 Layout Manager Reference Manual

### 4.1 Im\_arrange

SUMMARY	<p>Position child objects of frame in an imaginary grid. Before calling this function the properties of the child objects must be specified using <code>lm_linkobj</code>. As explained in the previous chapter. The position of each object is determined by:</p> <ul style="list-style-type: none"> <li>• the position of its parent frame;</li> <li>• its <code>normpos</code> property;</li> <li>• its <code>pixelpos</code> property.</li> </ul> <p>You may specify these properties separately for each object, but it is far more efficient to let <code>lm_arrange</code> do this for you. A typical call to <code>lm_arrange</code> looks like this:</p> <pre>hframe=lm_createframe(...) for row=1:4     for col=1:3         h(row,col)=uicontrol(...)     end end lm_linkobj(h,hframe) lm_arrange(hframe) lm_resize;</pre> <p><code>lm_arrange</code> places each object in a cell of an imaginary grid. Therefore each object must have a "row" and "col" property specified. In the example above these properties are set in the call to <code>lm_linkobj</code>: this function assigns "row=i,col=j" to any nonzero element <code>h(i,j)</code>.</p> <p>Below you will find information on how to merge cells to span multiple columns or rows.</p>
CALL	<pre>[size,grid]=lm_arrange(hframe,property,value,...) [size,grid]=lm_arrange(hframe,propertystruct)</pre>

INPUT	hframe	Frame object. This is a handle to an object that is created with a call to <code>lm_createframe</code> .	
	<b>Property</b>	<b>Default</b>	<b>Purpose</b>
	LMAR	10	Margin <i>left</i> between grid and innerframe (all margins are in pixels).
	RMAR	10	Margin <i>right</i> between grid and innerframe.
	TMAR	15	<i>Top</i> Margin between grid and innerframe.
	BMAR	6	Bottom margin between grid and innerframe.
	HMAR	5	Horizontal margins between columns of grid. HMAR may be specified as a scalar or as a vector.
	VMAR	1	Margins between rows. VMAR may be specified as a scalar or vector.
	PIXELW		Overrule width of innermargins of frame with this value. By default this value is computed by <code>lm_larrange</code> based on the margins and the contents of the grid.
	PIXELH		Overrule height of innermargins of frame with this value. By default this value is computed by <code>lm_arrange</code> based on the margins and the contents of the grid.
	NORESIZE	false	If true: Do not compute and set frame size based on its contents. The default value implies that <code>lm_arrange</code> computes the size of the frame.
	HEQUAL	false	If true: After computing the width of each <i>column</i> (both in pixels and normalized coordinates) change these attributes to their maximum values. The effect is that all columns have equal width.
	VEQUAL	false	If true: After computing the height of each <i>row</i> (both in pixels and normalized coordinates) change these attributes to their maximum values. The effect is that all rows have equal height.
	HNORM	false	If true: Assign normalized width to each column. If false: Ignore normalized properties for column width.
	VNORM	false	If true: Assign normalized width to each row. If false: Ignore normalized properties for row height.
HCENTER	0	This option is used to align objects within a frame. The property applies to all children that are aligned through a call by <code>lm_arrange</code> . Note: this option is only used in rare occasions. if 0: left align if 1: center items in horizontal direction if 2: right align items in horizontal direction NOTE: if HNORM==1 the HCENTER option is ignored	
VCENTER	0	This option is used to align objects within a frame. The property applies to all children that are aligned through a call by <code>lm_arrange</code> . Note: this option is only used in rare occasions. if 0: top align if 1: center items in vertical direction if 2: bottom align NOTE: if VNORM==1 the VCENTER option is ignored.	

<p>INDIRECT INPUT</p>		<p>Apart from the attributes above that are passed on to <b>lm_arrange</b> directly, <b>lm_arrange</b> checks a number of properties of the uicontrol objects that are positioned using <b>lm_arrange</b>. These properties are stored in the <i>application data</i> of each object. These objects are set by <b>lm_linkobj</b> for a group of objects or by <b>setappdata</b> for a single object.</p> <p>Important! These properties should be set <i>after</i> the call to <b>lm_linkobj</b> but <i>prior</i> to the call to <b>lm_arrange</b> using the <b>setappdata</b> command. This command adds these properties to the application data of each object.</p>
	<p>row,col</p>	<p>These properties tell <b>lm_arrange</b> in which cell the object goes. Usually these properties are set by <b>lm_linkobj</b>. Objects that do not have a row or col property will be ignored by <b>lm_arrange</b>.</p> <p>Normally the row and col property are set in a call to <b>lm_linkobj(h,h_frame)</b>. Typically h represents a matrix of handles. Element h(i,j) will receive the properties "row=i" and "col=j".</p> <p>If you need to make the component span multiple cells you must pass multiple rows or columns to a cell. This cannot be done by a vectorized call to <b>lm_linkobj</b>. Instead <b>setappdata</b> needs to be called separately for the cell or cells that are joined .</p> <p>Example (multiple cell spanning)</p> <pre>lm_linkobj(h,h_frame) ; %currentlty "col=1" is assigned to h(1,1) setappdata(h(1,1),'col',[1:3])</pre>
	<p>pixelpos</p>	<p>Pixelpos is a 4 element vector [x,y,w,h] that tells <b>lm_resize</b> how to compute the position of an object. By default, the function <b>lm_arrange</b> will set this property by looking at the <i>extent</i> of each object. The property pixelpos may be set prior to a call to <b>lm_arrange</b>. For example:</p> <pre>setappdata(h(1,1),'pixelpos',[0 0 200 20]);</pre> <p>In this case will not use the extent of h(1,1) but instead it will use the width and height components 200 and 20. The x and y position (in this case [0,0]) will be be overruled by <b>lm_arrange</b>.</p>
	<p>normpos</p>	<p>Normpos is a 4 element vector [xn,vn,wn,hn] that tells <b>lm_resize</b> the normalized position of an object.</p> <p>If the flags HNORM en VNORM are passed to <b>lm_arrange</b>, this function will assign these property to each object. Otherwise the normpos property will be reset to [0 0 0 0].</p> <p>Unlike pixelpos, normpos cannot be computed from the extent of an object. You need to set this property for at least one element.</p>

OUTPUT	size=[w,h]	The width and height of the virtual grid including the margins. By default the flag NORESIZE is set to false. In this case a call to <code>lm_arrange</code> will affect the <code>pixelsize</code> attribute of a frame in such a way that it equals this computed size, increased with the margins set by the <code>minmarges</code> attribute. Normally you do not need this output.
	<pre>grid +----x     +----pixelpos     +----normpos +----y      +----pixelpos      +----normpos</pre>	Coordinates of the grid raster as computed by <code>lm_arrange</code> . Normally you do not need this output.

### Troubleshooting

If you are new to the Layout Manager, you might encounter situations where the results are different from what you expect. To facilitate troubleshooting, the table below lists a few typical examples where the result might be different from what one may expect initially and the reason for this.

Desired result	Normalize column 1 use both columns for header object.
Code example	<pre>lm_linkobj(h,hframe); setappdata(h(1,1),'col',[1,2]); setappdata(h(1,1),'normpos',[0 0 1 0]); lm_arrange(hframe,'VMARGE',1,'HMARGE',5,'TMARGE',0,'HNORM',1);</pre>
Result/Problem	First column is not normalized. Header object not distributed.
Explanation	Because the first object is spans 2 columns, its <code>normpos</code> property is not used!
Remedy	<code>setappdata(h(2,1),'normpos',[0 0 1 0]);</code>

Desired result	Use column 1 and 2.
Code example	<pre>lm_linkobj(h,hframe); setappdata(hh,'col',1:2); lm_arrange(hframe,'VMARGE',1,'HMARGE',5,'TMARGE',0,'HNORM',1);</pre>
Result/Problem	Uicontrol appears only in column 2.
Explanation	Error: Joint space in column 1 and 2 is too narrow, combined with right aligned text. This makes it look like everything is displayed in column 2 but it is not.
Remedy	Make more space available for column1 and 2.

Desired result	Plot line in grid using <code>lm_arrange</code> .
Code example	<pre>hh=line; setappdata(hh,'pixelpos',[0 0 30 0]);</pre>
Result/Problem	Line exceeds gridcell.
Explanation	<code>lm_resize</code> interprets <code>normpos</code> and <code>pixelpos</code> as <code>[x1 y1 x2 y2 x3 y3 ...]</code> <code>lm_arrange</code> sets them to <code>[xll yll width height]</code> .
Remedy	set <code>pixelpos</code> separately after <code>lm_arrange</code> is called.

Desired result	Make object N pixels high.
Code example	<pre>h(r,c)=uicontrol('style','list',... lm_linkobj(f,h_frame); setappdata(h(r,c),'pixelpos',[0 0 N 0]); lm_arrange(h_frame);</pre>
Result/Problem	Object height does not change.
Explanation	Setting element 3 of <code>pixelpos</code> (this element sets the width).
Remedy	Setting element 4 of <code>pixelpos</code> (this element sets the height).

Desired result	Object spans full width of frame.
Code example	<pre>h=uicontrol('style','list') lm_linkobj(h,hframe,'row',1,'col',1); %use full width : setappdata(h,'normpos',[0 0 1 0]); %set fixed height of object : setappdata(h,'pixelpos',[0 0 0 200]); lm_arrange(hfr2,'TMARGE',0,'LMARGE',10,'RMARGE',10); lm_resize;</pre>
Result/Problem	Object does not align with right side of frame.
Remedy	Add "'HNORM',1" in de <code>lm_arrange</code> call.

Desired result	Header spans multiple columns. The columns should not be resized based on the header. The header should not be clipped to the column it is defined in  Example: <pre>Timerange from to</pre>
Code example	<pre>h(1,5)=uicontrol('style','text','Timerange') setappdata(h(1,5),'ignorew',true);</pre>
Result/Problem	The header is left aligned but there is insufficient room for the header.
Remedy	set both the "ignorew" and the "keeppixelsize" flag:  <pre>%header should not impact column width setappdata(h(1,5),'ignorew',true); %column width should not impact header width setappdata(h(1,5),'keeppixelsize',true);</pre>

## 4.2 lm\_childframes

SUMMARY	List the child-frames directly below a given frame.	
CALL	<code>h_frames = lm_childframes(hframe)</code>	
INPUT	<code>hframe</code>	Handle of parent frame (scalar).
OUTPUT	<code>h_frames</code>	List of handles of child frames.

## 4.3 lm\_createframe

SUMMARY	Create a frame. A frame is a visible or invisible panel that is a parent of other frames, or interface components (or both).	
CALL	<pre>hframe = lm_createframe(property,value,...) hframe = lm_createframe(hparent,property,value,...)</pre>	

INPUT	hparent	Handle to parent. This may be another frame or a figure object. If no handle is specied <code>lm_createframe</code> uses <code>gcf</code> as the parent object.
	property/value pairs	
		Note: Where possible the Layout Manager uses "true" or "false" for values where on basis of analogy with Matlab commands one may have expected "on" or "off". This has been done for reasons of computational efficiency.
	active	Default: true. Visibility of this frame <i>and</i> its children. Setting this value to <i>false</i> tells <code>lm_resize</code> to hide this frame and all its children.
	border	Default: true. If set to true the frame will be visualized using the <code>uicontrol/frame</code> object that also stores the frame properties. In many cases one may not want to visualize the frame this way. For example when a frame contains an axes this will be obscured by <code>uicontrol</code> objects. The border will be displayed on the innermargins of the frame (see Figure 10).
	enable	This property controls the enabled status of all child objects enabled=0 → display child object with enabled status="off" enabled=1 → display child object with enabled status="on"  Note: you may overrule the frame enables status by setting the "enable" property of individual objects, see <code>lm_linkobj</code> .
	lineprops	This property is similar to border. If you set this value to <code>lm_lineprops</code> a shadowed border will be plotted. This border is plotted using objects of type "line". Unlike <code>uicontrol</code> objects, these will not obscure axes objects. You may pass arguments to <code>lm_lineprops</code> to control the appearance of the border. Advanced users may replace <code>lm_lineprops</code> with a new function for even more control of the border appearance.  Example: <code>lm_createframe('lineprops',lm_lineprops);</code>
	patchprops	This property can be used instead of "lineprops", and creates a patch rather than a line object to display the frame. This may be useful for controlling the background color of a frame without obscuring any axes objects.  Example: <code>lm_createframe('patch',lm_patchprops('facec',[1 1 1]))</code>
	minmarges	<code>minmarges=[left,bottom,right,top]</code> and defines the coordinate system in which child objects of a frame are plotted (see Figure 10). Also the border of a frame is plotted on the innerframe.
	title	If a title is specified, it is displayed using a <code>uicontrol</code> object on the left top of a frame.
	pixelsize	Size of the frame in pixels. This is a 2-element vector. You may set this vector to <code>[NaN NaN]</code> . In this case the frame size will computed from the sizes of the child frames.
	normsize	Normalized size of the frame. The space that is left after adding up the <code>pixelsize</code> properties of all child frames is distributed proportional to the <code>normsize</code> property.
	maxpixelsize	If <code>normsize</code> is nonzero, the displayed frame size may be higher than the value in <code>pixelsize</code> . However, the size of a frame will not exceed the values specified in <code>maxpixelsize</code> . This applies separately to both dimensions.
normposition	Position of top frame relative to the figure in normalized coordinates. This property only applies when the parent of a frame	



		you wish to separate.
	modeValue	This must be one of the following: <ul style="list-style-type: none"> <li>• “proportional”: change the size of all frames when moving the divider;</li> <li>• “neighbour”: only change the size of neighbouring frames when moving the divider.</li> </ul>
OUTPUT	Usually the output arguments of this function are not needed.	
	hframe	Frame that holds the divider.
	jseparator	Handle to the divider object.
REMARK	This function requires the <i>Modelit User Interface Components Toolbox for Matlab</i> for creating the divider object.	

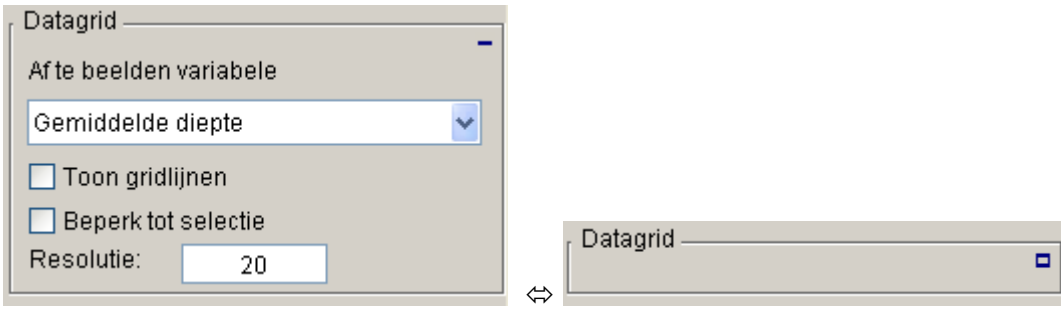
EXAMPLE

Figure 11: Draggable dividers may be used to separate frames

#### 4.7 Im\_doubleframe

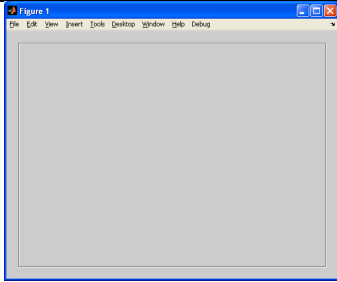
SUMMARY	Create a frame that can be minimized to a place holder. A call to <b>lm_doubleframe</b> is equivalent to 3 calls to <b>lm_createframe</b> and creates the following hierarchy: frame container +----regular frame (component frame 1) +----minimized frame (component frame 2)
CALL	[h_ItemFrame,h_frame]=... lm_doubleframe(h_parent,titlestr,outer_frame_opt,inner_frame_opt)



INPUT	<code>h_parent</code>	parent frame of the frame container.
	<code>titlestr</code>	title of the frame container.
	<code>outer_frame_opt</code>	cell array that contains the arguments to be passed on in the call to <code>lm_createframe</code> for the frame container. These will be appended to the default properties: {'normsize',[1 0],... 'pixelsize',[0 NaN],... 'border',0,... 'splithor',0}
	<code>inner_frame_opt</code>	cell array that contains the arguments to be passed on in the call to the component container. These will be appended to the default properties: {'normsize',[1 1],... 'lineprops',lm_lineprops,... 'active',1}
OUTPUT	<code>h_ItemFrame</code>	handle of the container frame. This is the frame that holds the objects.
	<code>h_frame</code>	handle of the frame container.
SEE ALSO	<code>lm_framelist</code>	
EXAMPLE	 <p><b>Figure 12:</b> A frame that is created using the <code>lm_doubleframe</code> command. The frame contains a button that toggles between minimized and normal view.</p>	

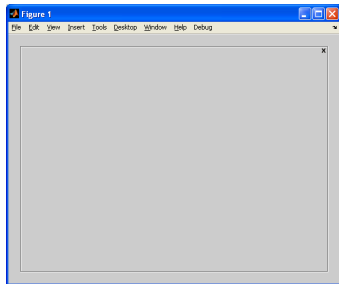
#### 4.8 `lm_exitbutton`

SUMMARY	Add an exit button to a frame. By default this button is placed in the upper right corner of a frame.	
CALL	<code>h = lm_exitbutton(hparent, BACKG, callback)</code>	
INPUT	<code>hparent</code>	Handle of parent frame.
	<code>BACKG</code>	Color for transparant part of button.
	<code>callback</code>	Additional function to call when frame is closed.
OUTPUT	<code>h</code>	Handle of the uicontrol that is used as exitbutton.
SEE ALSO	<code>lm_exittext</code> , <code>lm_frameonoff</code>	
NOTE	In many cases the function <code>lm_exittext</code> is preferred as this function prints an "x" symbol with a callback rather than a button.	
EXAMPLE	<pre>h1 = lm_createframe(gcf); h2 = lm_createframe(h1, 'minmarges',[20 20 20 20],...     'lineprops',lm_lineprops); lm_resize;</pre>	



**Figure 13:** Frame before adding exit button

```
lm_exitbutton(h2);
lm_resize;
```



**Figure 14:** Frame after adding exit button

#### 4.9 Im\_enableonoff

SUMMARY	Toggle enabled status of frame.	
CALL	Enable = lm_enableonoff(option, resize, handle)	
INPUT	option	Imposed enabled status, possible values: on, off or toggle(default).
	resize	Call <b>lm_resize</b> when done, possible values: true (default) or false.
	handle	Handle of object that holds framehandle as its userdata. This handle is used to retrieve the handle of the frame that is enabled or disabled.
OUTPUT	enable	Resulting enabled status (0 or 1).

#### 4.10 Im\_exittext

SUMMARY	This function is similar to <b>lm_exitbutton</b> but instead of using a uicontrol-pushbutton, it prints an "x" in the upperright-corner of the frame. This is convenient when the frame is fully occupied by an axes that would have been obscured by a button, but not by a text symbol.	
CALL	lm_exittext(h_frame,BDCALL,align,str)	
INPUT	h_frame	Frame handle.
	BDCALL	Function to call when button is pressed.
	align	Set this parameter to 1 to align the text on the bottom of the frame. Defaults value is 0.
	str	String to display. Defaults to "x". Other values to consider: "close", "cancel" etcetera.
OUTPUT	This function does not return any output arguments.	

### 4.11 lm\_framelist

SUMMARY	<p>This function creates a container for a collection of frames that are arranged vertically. If the total height of the frame exceeds the available space, the frames may be moved using a slider. The function hence constitutes of a viewport for its vertically arranged child frames. A call to <code>lm_framelist</code> creates the following frame hierarchy:</p> <pre> frame container +----viewport frame (handle exported by this function)     +---- frames to be added afterwards     +----     +---- +----slider frame (this frame contains a slider that                     is linked to the viewport)                 </pre>	
CALL	<code>[h_inner,h_outer] = lm_framelist(hparent,innerOpt,outerOpt)</code>	
INPUT	<code>hparent</code>	Parent frame for frame container
	<code>innerOpt</code>	Extra options for call to <code>lm_createframe</code> that creates inner frame (viewport frame). These options should be listed in a cell array.
	<code>outerOpt</code>	Extra options for call to <code>lm_createframe</code> that creates outer frame (frame container). These options should be listed in a cell array.
OUTPUT	<code>h_inner</code>	Handle of inner frame (viewport frame). This handle should be used when adding new frames.
	<code>h_outer</code>	Handle of outer frame. This handle can be used to set the active property of the outer frame.
EXAMPLE	<pre> h_inner = ... lm_framelist(hparent,{'tag','innerTag'}, {'tag','outerTag'}); define_subframe1(h_inner); %define children define_subframe2(h_inner); define_subframe3(h_inner); define_logo(h_inner);                 </pre>	
SEE ALSO	<code>lm_doubleframe</code>	

### 4.12 lm\_frameonoff

SUMMARY	<p>This function is usually specified as a callback of a button in the toolbar that switches a specific frame on- and off. The handle of the frame that must be toggled must be stored in the userdata of the button.</p>	
CALL	<code>active = lm_frameonoff(option,resize,obj)</code>	
INPUT	<code>option</code>	String with possible values: <ul style="list-style-type: none"> <li>• “toggle” (default) → toggle active status of frame;</li> <li>• “on” → show frame;</li> <li>• “off” → hide frame.</li> </ul>
	<code>resize</code>	Boolean, <ul style="list-style-type: none"> <li>1 (default) → call resize function immediately.</li> <li>0 → suspend resize function.</li> </ul>
	<code>obj</code>	Handle of object that stores framehandle in its userdata. Defaults to “gcbo”.
OUTPUT	<code>active</code>	Boolean (usually not needed) <ul style="list-style-type: none"> <li>true → the frame was made visible.</li> <li>false → the frame was hidden.</li> </ul>
SEE ALSO	<code>lm_enableonoff</code>	

### 4.13 lm\_initaxes

SUMMARY	A call to this function sets up an invisible axes that is used to position "line", "patch", "image" and "text" objects that are positioned by the Layout Manager.	
CALL	<code>hax = lm_initaxis(HWIN,LAYER)</code>	
INPUT	HWIN	Handle of the window for which pixel axes will be set (defaults to gcf).
	LAYER	Layer number. If needed, multiple axes objects can be created to enable plotting in different layers. Frames plotted in the current axes obscure lines and text objects in other layers. <code>lm_initaxes</code> checks if an axes for this layer has already been set up. If so the function does not create a new axes, so there is no harm in calling the function more than once. Default value is 1.
OUTPUT	hax	Handle to the axes object that has been created.
EXAMPLE	<pre>hax=lm_initaxes; h=text(1,1,'my text','parent',hax); lm_linkobj(h,hframe,'pixelpos',[ 10 10 20 20]); lm_resize;</pre>	

### 4.14 lm\_innerpixelsize

SUMMARY	Change pixelsize property of frame so that the size of the innerframe matches a given size. This utility is useful if the size of what goes into the frame is known and one wants to shrink the outer frame so that it exactly fits its contents. The resulting pixelsize depends on the specified inner-size and the property "minmarges" that is stored in the frame. See also Figure 10.	
CALL	<code>Outbordersize = lm_innerpixelsize(hframe, innerborderpixelsize)</code>	
INPUT	hframe	Handle of parent frame.
	innerborderpixelsize	[Width,Height] 2 element vector containing the size of the innerframe, in pixels.
OUTPUT	outbordersize	[Width,Height] Computed pixelsize.
EXAMPLE	For an example, see the source code of <code>lm_arrange</code> . <code>lm_arrange</code> calls <code>lm_innerpixelsize</code> after the dimensions of the frame are computed.	

### 4.15 lm\_isparent

SUMMARY	Find out if a given frame is child of any of a list of candidate parent frames	
CALL	<code>istrue = lm_isparent(h, hframes)</code>	
INPUT	h	Frame handle (scalar).
	hframes	Handles of potential parent frames.
OUTPUT	istrue	Boolean, true if any of hframes is the parent of h, false otherwise.

### 4.16 lm\_lineprops

SUMMARY	This function returns a structure that can be passed to the Matlab "line" command. If called without arguments it will produce the settings that are needed to plot a "standard" border. Any property value pair that can be passed to the line command can also be passed to <code>lm_lineprops</code> . Additionally the argument "shadowed" may be passed. This argument tells the Layout Manager to plot not one, but two lines. This results in a shadow effect.	
CALL	<code>s = lm_lineprops(property, value,...)</code>	
INPUT	property,value	Any line property.

	'shadowed', B	Boolean, B==1 → apply shadow. B==0 → do not apply shadow.
OUTPUT	s	A structure that can be passed to the Matlab "line" command (after stripping "Shadowed" from this structure). Any specified property-value pair will overrule or add a field in this structure.  s +----XData (double) +----YData (double) +----Color (double array) +----HitTest (char array) +----LineWidth (double) +----Shadowed (double)
SEE ALSO	<b>lm_patchprops</b>	

#### 4.17 lm\_linkobj

SUMMARY	This function registers an object as a child of a frame that was created earlier using <b>lm_createframe</b> . At the same time a number of properties can be set. It is also possible to set properties individually using the Matlab <b>setappdata</b> command. <b>lm_linkobj</b> may be called for a single object handle or for a matrix of object handles. In this case <b>lm_linkobj</b> sets the "row" and "col" properties as used by <b>lm_arrange</b> .	
CALL	<code>lm_linkobj(h,hframe,property,value,...)</code>	
INPUT	h	Handle of object or matrix of object handles. If h is a matrix the row and col attribute will be set by <b>lm_linkobj</b> :  <code>lm_linkobj(h,hframe)</code>  is equivalent to:  <pre>for row=1:size(hobj,1)   for col=1:size(hobj,2)     if hobj(row,col)       lm_linkobj(h(row,col),hframe,...                 'row',row,'col',col);     end   end end</pre>
	hframe	Handle of parent frame.
	<i>property,value</i>	Property value pairs. The allowed properties are listed below. Property names are not case-sensitive and it suffices to specify the first letters of the property name, as long as these letters uniquely identify the property name.
	row,col	Properties that are used by <b>lm_arrange</b> . Specifying these properties and calling <b>lm_arrange</b> afterwards will cause <b>lm_arrange</b> to fill-in the <code>pixelpos</code> and <code>normpos</code> properties. Generally this is the fastest way to arrange objects.
	normpos pixelpos	Normpos and pixelpos are 4 element vectors <code>[xn,vn,wn,hn]</code> and <code>[xp,vp,wp,hp]</code> that tells <b>lm_resize</b> the normalized and pixel position of an object in the following way:  Consider Figure 15 below.  Suppose the position of the parent frame is:

		<pre>[X, Y, W, H]</pre> <p>and the minmarges property is set to: [left, low, right, top]</p> <p>then the position of the inner frame is given by: [Xi, Yi, Wi, Hi]= [X+left, Y+low, W-left-right, H-low-top]</p> <p>The object position is computed as follows:</p> <pre>objectpos=[x, y, w, h] x=Xi + xn*Wi + xp y=Yi + yn*Hi + yp w=wn*Wi + wp h=hn*Hi + hp</pre>
	<p>visible</p>	<p>Boolean value. 0 → hide object 1 → show object</p> <p>It may seem strange that this property is specified because handle graphic objects already have a property "visible". However in the Layout Manager the visible property is also controlled by the active status of the frame. Therefore this extra attribute is needed.</p> <p>The dynamic (handle graphic) property "visible" is evaluated as follows by <code>lm_arrange</code>:</p> <pre>object <input type="text"/> property frame property visible  0 1        0 'off' 'off'        1 'off' 'on'</pre>
	<p>enable</p>	<p>Like the visible property, the enable property may also be controlled at frame level. In order to be able to override properties set at frame level, the enable attribute is available at object level.</p> <p>The dynamic (handle graphic) property "enable" is evaluated as follows by <code>lm_arrange</code>:</p>

		<pre> object property frame property enable  0 1  0 'off' 'off'  1 'off' 'on'  2 'off' 'inactive'  3 'off' 'off'  4 'on' 'on'  5 'inactive' 'inactive' </pre>
	clipping	Boolean, If this property is set to "true" and property "clipframe" contains a frame handle, objects that appear outside the clipframe will be clipped.
	clipframe	Frame handle, this property works together with the "clipping" property.
	keeppixelsize	This property works together with the function <code>lm_arrange</code> . When this flag is set, <code>lm_arrange</code> will not change the 3rd and 4rd component of the <code>pixelpos</code> property. See also the documentation of <code>lm_arrange</code> .

	frame. Interactively changing the slider position changes the frame property "verticalshift" and subsequently calls <code>lm_arrange</code> . The result is that frame contents are moved up and down, hence creating the effect of a viewport. If for some reason an object needs to be excluded from this effect you may specify the property "keepypos=true". An example where this is useful is an exit button in the UR corner of a frame.
OUTPUT	This function sets the application data of an object and a frame, but does not return any output arguments.

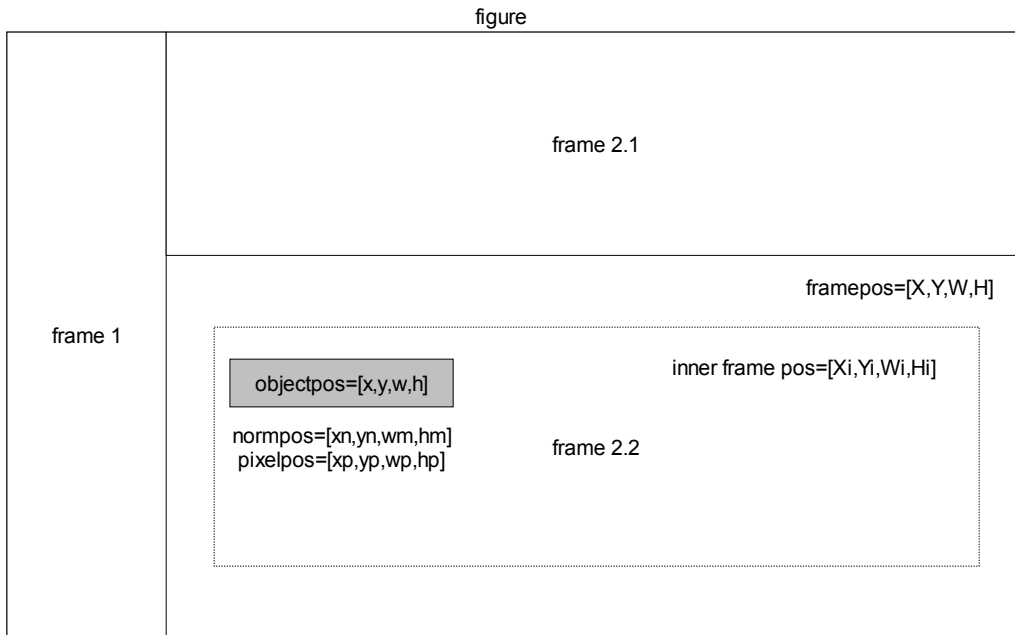


Figure 15: Overview of parameters with an impact on object position

#### 4.18 lm\_linkslider2frame

SUMMARY	Make y-position of frame content dependent on a vertical slider.	
CALL	<code>lm_linkslider2frame(hslid, targetframe)</code>	
INPUT	<code>hslid</code>	Handle of uicontrol of style "slider".
	<code>targetframe</code>	Handle of target frame. The contents of this frame can be moved by using the slider.
OUTPUT	No direct output. The slider handle is stored in the target frame in the property "slider".	

#### 4.19 lm\_listframeHandles

SUMMARY	Retrieve frame handles and frame data for the specified figure.	
CALL	<code>[FrameHandles, FrameData] = lm_listframeHandles(hfig)</code>	
INPUT	<code>hfig</code>	Figure handle (defaults to <code>gcf</code> ).
OUTPUT	<code>FrameHandles</code>	Nx1 list of frame handles.
	<code>FrameData</code>	Nx1 struct array with corresponding application data.



#### 4.20 `lm_parentframe`

SUMMARY	Find out to which frame the object with handle <code>h</code> is linked.	
CALL	<code>hframe = lm_parentframe(h)</code>	
INPUT	<code>h</code>	object handle (scalar).
OUTPUT	<code>hframe</code>	handle of parent frame, empty if object is not linked to a frame.

#### 4.21 `lm_patchprops`

SUMMARY	This function returns a structure that can be passed to the Matlab "patch" command. If called without arguments it will produce the settings that are needed to plot a "standard" transparent patch. Any property-value pair that can be passed to the "patch" command can also be passed to <code>lm_patchprops</code> .	
CALL	<code>s = lm_patchprops(property, value, ...)</code>	
INPUT	<code>property,value</code>	Any Matlab "patch" property.
OUTPUT	<code>s</code>	A structure that can be passed to the Matlab "patch" command. Any specified property value pair will overrule or add a field in this structure.  <pre>s +----XData (double) +----YData (double) +----FaceColor (char array) +----HitTest (char array) +----LineWidth (double)</pre>
SEE ALSO	<code>patch</code> , <code>lm_lineprops</code>	

#### 4.22 `lm_pixelsize`

SUMMARY	Get pixelsize of frame.	
CALL	<code>pixelsize = lm_pixelsize(hframe)</code>	
INPUT	<code>hframe</code>	Frame handle.
OUTPUT	<code>pixelsize</code>	Vector [height, width] with the pixelsize of the frame.

#### 4.23 `lm_resize`

SUMMARY	Resize the figure and (re)position all the objects it contains.	
CALL	<code>lm_resize(hfig, event)</code>	
INPUT	<code>hfig</code>	Handle of the figure to be resized.
	<code>event</code>	Standard Matlab callback argument, not used. This argument makes it possible to define <code>lm_resize</code> as a callback function.
OUTPUT	All frames created with <code>lm_createframe</code> and all the objects linked to these frames with <code>lm_linkobj</code> are (re)positioned in the figure.	

#### 4.24 `lm_shiftrank`

SUMMARY	Change the rank of a frame. If the "splithor" property of the parent frame is set to true this causes the frame to move to the left or right, or up or down if this property is set to false.	
CALL	<code>hchange = mbdshiftrank(option, resize)</code>	

INPUT	option	String with shift direction, possible values: <ul style="list-style-type: none"> <li>• “up” increase rank of frame;</li> <li>• “down” decrease rank of frame.</li> </ul>
	resize	Boolean, if true call <code>lm_resize</code> afterwards
OUTPUT	hchange	Handles of the frames whose position has changed.
REMARK	The handle of the frame for which the rank must be changed must be stored in the userdata field of the uicontrol whose callback is executed.	

#### 4.25 lm\_sortframes

SUMMARY	Create a sorted list of frames which are create with <code>lm_createframe</code> , the frames are sorted based on level in the hierarchy, parent and rank.	
CALL	<code>[FrameData, parentIndex] = lm_sortframes(hfig)</code>	
INPUT	hfig	Figure handle.
OUTPUT	FrameData	Structarray with collected information per frame <ul style="list-style-type: none"> <li>+----stack[]: debug information</li> <li>  +----file (char array)</li> <li>  +----name (char array)</li> <li>  +----line (double)</li> <li>+----treetop (logical)</li> <li>+----parenthandle (double)</li> <li>+----rank (double)</li> <li>+----normsize (double array)</li> <li>+----pixelsize (double array)</li> <li>+----maxpixelsize (double array)</li> <li>+----normposition (double array)</li> <li>+----pixelposition (double array)</li> <li>+----enable (logical)</li> <li>+----splithor (double)</li> <li>+----border (double)</li> <li>+----exitbutton (logical)</li> <li>+----exitfunction (char)</li> <li>+----active (logical)</li> <li>+----exitbuttonhandle (double)</li> <li>+----minmarges (double array)</li> <li>+----children (double)</li> <li>+----textchildren (double)</li> <li>+----javachildren (double)</li> <li>+----uichildren (double)</li> <li>+----slider (double)</li> <li>+----patchhandle (double)</li> <li>+----linehandle (double)</li> <li>+----shadowlinehandle (double)</li> <li>+----level (double)</li> <li>+----showslider (double)</li> <li>+----handle (double)</li> <li>+----inborderpos (double)</li> <li>+----outborderpos (double)</li> <li>+----activenode (double)</li> <li>+----enablednode (logical)</li> </ul>
	parentIndex	List with the parent indices corresponding to each element in FrameData.

**4.26 lm\_title**

SUMMARY	Set or get the title of a frame.	
CALL	<pre>h = lm_title(hframe) h = lm_title(hframe, str, varargin)</pre>	
INPUT	hframe	Handle of frame.
	str	Title to be displayed in frame.
	varargin	Valid property-value pairs for a uicontrol of style "text".
OUTPUT	h	Handle to title object (uicontrol of style "text").