

What's new in pro Fit 5.5

This document is for experienced pro Fit 5.1 users, and provides a summary of the new features found in pro Fit 5.5, so that they can start to profit from the new version immediately.

Requirements and compatibility	2
System requirements	2
File compatibility	2
New features	3
New User Interface Experience and better integration with Mac OS 9.0	3
Date and Time data	3
Preview Window	3
Parameter Window	4
Predefined functions	4
Data Windows	4
Drawing Windows	4
Plotting	5
Preferences	6
Numerics Commands	6
Debug Window	8
More power for user-defined functions and programs	10
Input and Output	10
Date and time management	10
Mathematical operators, functions, and procedures	10
Complex data types: Matrices and Vectors	11
Creating and Saving Text files	12
Strings	13
Data Processing	13
Tags: Sending and getting data from a program.	13
Plotting	15
Drawing	16
Working with control shapes	16
Getting and Setting “Properties” of various pro Fit objects	20
New Pre-defined Procedures & Functions	21

Requirements and compatibility

System requirements

pro Fit is supported on Mac OS[®] version 8.5 or later. pro Fit requires at least 2 MByte of free memory (if virtual memory is off it requires at least 4 MByte), however, for normal activities, 5 MBytes are suggested.

pro Fit 5.5 is a Power PC only application. There are no pro Fit 5.5 versions for older machines based on Motorola's 680x0 family of processors. For users interested in running a version of pro Fit on those machines, pro Fit 5.1 is still available. pro Fit 5.5 allows to save most of its documents in pro Fit 5.1 format.

File compatibility

pro Fit 5.5 can import all files from pro Fit 5.1, except preferences files and compiled programs or functions.

pro Fit 5.1 can read text, data and drawing files of pro Fit 5.5, with the following restrictions:

- Data files that contain automatically calculated columns may not be read properly. Date and time columns are read as “double” columns.
- Drawing files must be saved as “pro Fit 5.1” files
- Parameter sets that were generated by pro Fit 5.1 can be read by pro Fit 5.5, but not vice versa.
- Compiled functions must be recompiled.

New features

New User Interface Experience and better integration with Mac OS 9.0

- “Open” and “Save” dialog boxes now use the new “navigation manager” routines.
- Hold down the command-key and click on a window title to find out where the file belonging to the window is.
- Click the document icon appearing to the left of a window title and drag it anywhere you want in a Finder window to move your document to a new location.
- Live feedback when resizing columns and info field in data windows
- Proportional scrollbars
- Text and data files now store the current selection
- All windows and tool palettes use the current system appearance.
- The status window appearing during fitting is now resizeable when it contains a list of parameters.
- All Dialog boxes and Alert boxes are movable.
- The “Check for update” button in the “about pro Fit” box can be used to find out if a newer version of pro Fit is available.

Date and Time data

pro Fit 5.5 understands and works with time data, *i.e.* absolute calendar dates and relative time. There are new column formats in data windows for date&time and relative-time data, and date and time data can be used in data-transforms, graphs, and by user-defined programs and functions.

Technical information

The Mac OS stores dates as the number of seconds since January 1, 1904 (for the technically minded, the date is stored as an integer number, 8 byte long).

pro Fit uses the same convention as the Mac OS to store dates, but uses “double” floating point values instead of integers. With this number representation, pro Fit can store and recognize dates with second precisions until up to 10^{15} (this corresponds more or less to a 6 byte long integer) seconds after January 1, 1904. pro Fit can store dates with second-precision up to **31 million years in the future**, and it can store dates with day-of-the-week precision up to 3.1 **billion** years (3×10^9) in the future.

However, the date-time conversion routines currently available in Mac OS 9 only support dates up to 29'940 AD for date-to-string conversions, date-calculations, etc. Up to this limit, pro Fit can store dates with a precision of milliseconds, while it can store dates in the present with a precision of approximately a microsecond.

Preview Window

The preview window has been redesigned. It is now resizeable, and many more options, like the color of the function curve, its thickness, or the size of data points, are user-selectable.

Parameter Window

The parameter window has been redesigned, and now supports up to 128 parameters. It is now possible to paste some parameter values selectively into consecutive parameter value fields by selecting a parameter and pasting tab-separated values.

Predefined functions

The predefined function **Spline** has more options that can be set by clicking the “Spline Settings” button in the parameter window. There is a new possibility to define the spline curve by using function parameters to set the coordinates of the points on which it is based. This allows, *e.g.*, to fit a Spline function to some set of noisy points, in order to get a smooth guide-to-the-eyes curve. When doing this, be careful not to choose too many points for the Spline-definition.

Data Windows

- Column-calculations can be stored and made permanent. They will be executed automatically whenever data is changed in a data window. The instruction for calculating a “calculation-column” can be changed via the column format dialog box, or can be generated via the command "Data Transform".
- Text columns can be used for sorting
- Sorting keeps the order of rows when they correspond to equal values in the sort-column. This allows to sort by more than one column, by using the sort command multiple times.
- The index column in data windows can be set to the default x or y column.
- New data column types and formatting options for date & time, and relative time data.

Drawing Windows

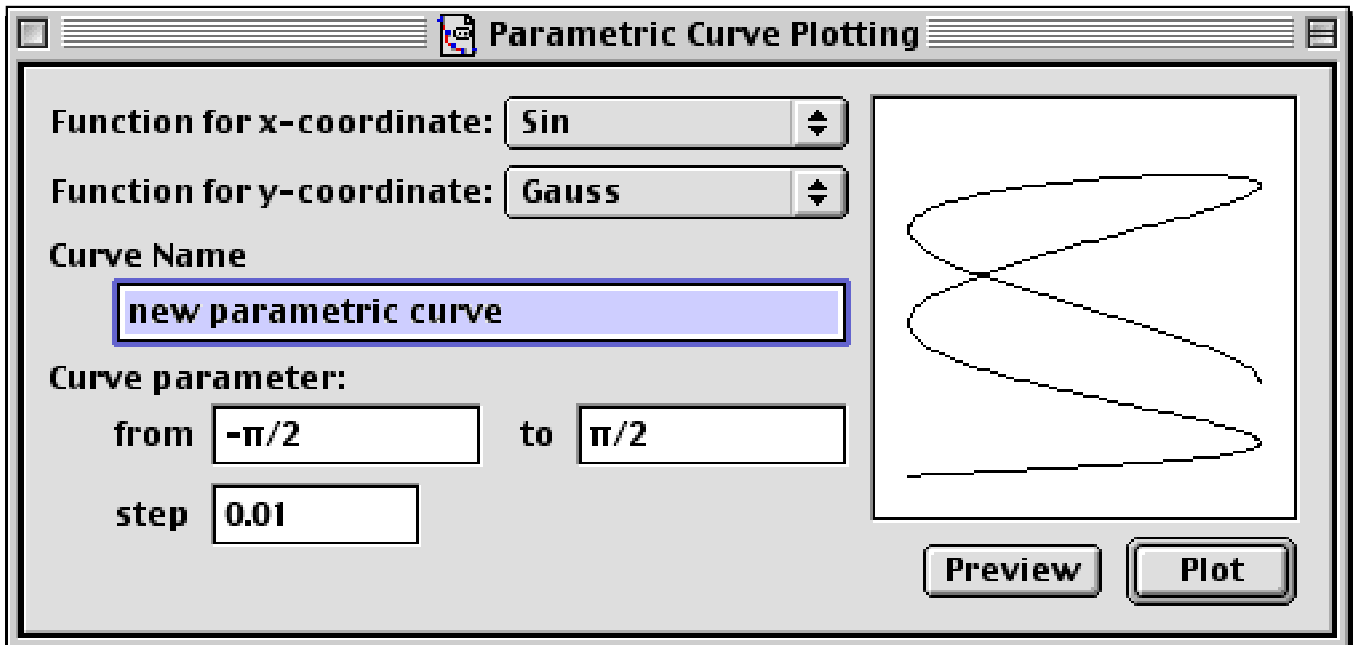
- New Save and Open possibilities for JPEG and PICT files, as well as saving of (uncompressed) GIF files. Anti-aliasing is supported when saving drawings as GIF or JPEG.
- Some small problems connected with EPS files have been fixed.
- Undo now restores the selection
- Command clicking for viewing graph coordinates now also works for grouped graphs.
- Lines, Polygons, and Rectangles can be rotated by any angle, not just multiples of 90 degrees. The “Coords” window displays the current angle, which can be edited to change the orientation of a shape. The Rotate Command in the Draw Menu has an “Other...” item that can be used to specify relative rotations of any number of degrees.
- Drawing shapes have properties that can be read and modified using the calls `GetShapeProperty` and `SetShapeProperties`, respectively. See below.

New drawing objects, or shapes

You can now create “control shapes” in drawing windows, such as buttons, checkboxes, radio buttons, popup menus, static texts and edit fields. These new control shapes can be used to create custom interfaces for user-defined programs by switching a drawing window from its normal state to its **dialog box state**.

This feature allows for a greater degree of customizeability for pro Fit. Developers and pro Fit users can now write useful additions to pro Fit (in the pro Fit definition language itself, or in an external module) with an appropriate and flexible dialog-box interface.

There are already several examples of the new kind of programs and plug-ins that are made possible by the new control-shapes of drawing windows. A simple example is the “Parametric Curve Plotting” dialog box shown below, which allows to plot parametric curves of the kind $(x = f(t), y = g(t))$ in the current graph (In the example below, $f = \text{Sin}$, $g = \text{Gauss}$).



In order to use a drawing window with control shapes as a dialog box, you must “attach” a program to the drawing window (see below). This program is called when somebody clicks or types into a control.

Controls can only be operated when the drawing window is in “dialog box” mode. To bring a drawing window into this mode, either choose “Get Info...” from the File menu and check “Is dialog window”, or click into the drawing window while holding down the control key and choose “Dialog Window” from the contextual menu that appears.

Control shapes (and any other shapes) can be accessed by a program using their **names**. To view and change the name of a drawing-shape, select the shape and choose “Shape Settings...” from the Draw menu. (You can also double-click most shapes for getting into the corresponding dialog box. When a window is in dialog mode, command-double-click a shape to change its settings.)

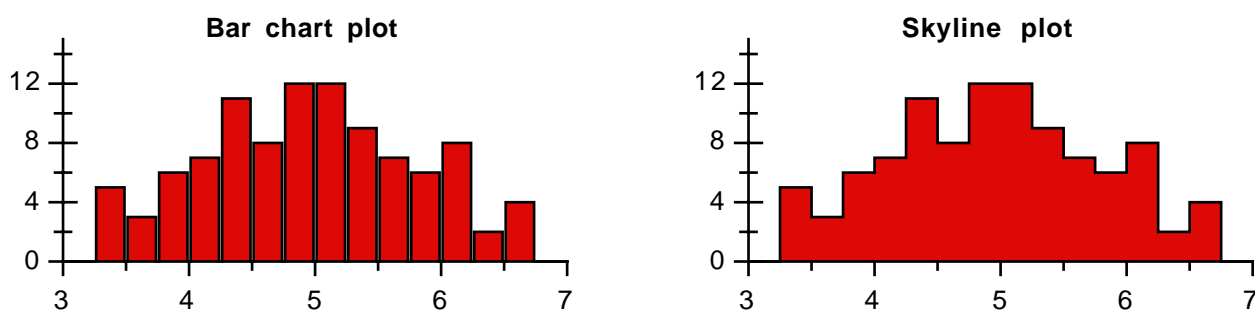
A detailed introduction to using control shapes is given below.

Plotting

- Bar chart and skyline plots.
- Graphs can now have rotated labels.
- Graph labels can be displayed as date and time.
- Log axes can now have 0, 1, 4 or 8 minor ticks.
- It is possible to set a percentage of the graph range (in a preferences panel) where points and function values are recorded, even if they are outside the graph ranges at the moment of plotting

Bar charts / Histograms

Pro Fit supports two more plotting types: Bar charts and skyline plots.



To use these two types, you choose “Plot Data...” from the “Draw” menu, and then select the desired plot type from the popup menu titled “Plot type”. Alternatively, you can change the type of an existing data plot by double-clicking the graph and choosing the “Curves” panel. Several options for bar charts and skyline plots can be set for each graph by double-clicking it and going to the “Bar Charts” panel.

Preferences

pro Fit now looks for/creates a “pro Fit Preferences” folder inside the system's preferences folder. In this folder it looks for a file “pro Fit 5.5 preferences” and creates one if no such file is found. If the “pro Fit Preferences” folder contains a folder named “pro Fit modules”, the modules in this folder are linked to pro Fit during start-up.

Various new options have been added to the preferences command:

Preview panel: You can now set more options for the size and color of the various items in the preview window.

Date & Time panel: Allows you to set various preferences for displaying dates and time.

Numerics Commands

Extrema

The Tabulate Extrema Menu command (in the submenu Analyze of the Menu Calc) now produces a data window which also lists the function value at the extremum, not only the x-coordinate of the extremum. Moreover, it differentiates between maxima and minima by putting them in two different columns

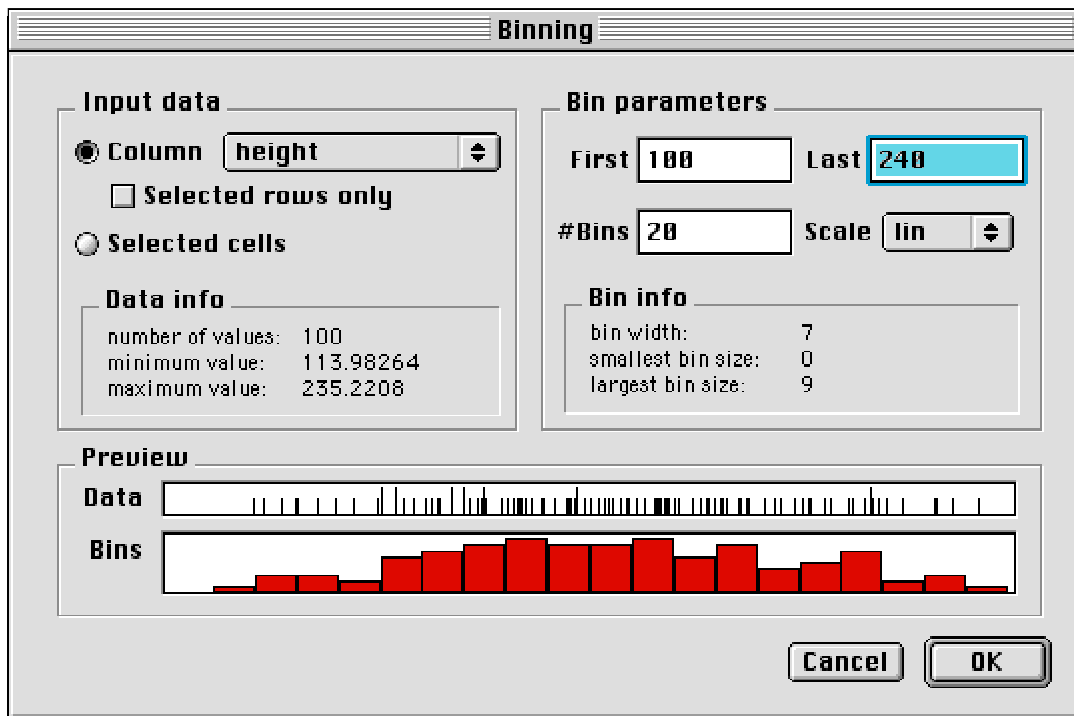
Binning

pro Fit can now “bin” data, a feature that is useful to create, for example, histograms representing the statistical distribution of some parameter inside a population. “Binning” is the process of putting data into bins, i.e. consecutive intervals. For each data value, the bin or interval it belongs to is determined and the size of the bin is increased by 1. In other words, the number of values in each bin are counted.

Example

You analyze the height of 1000 people. You put all height values into a data window. Now you want to plot a histogram, each histogram bar giving the number of people that have a height in a given 2 cm

interval. For this purpose, you choose “Binning” from the Calc menu, choose your data, define bins of 2 cm width, and click OK.



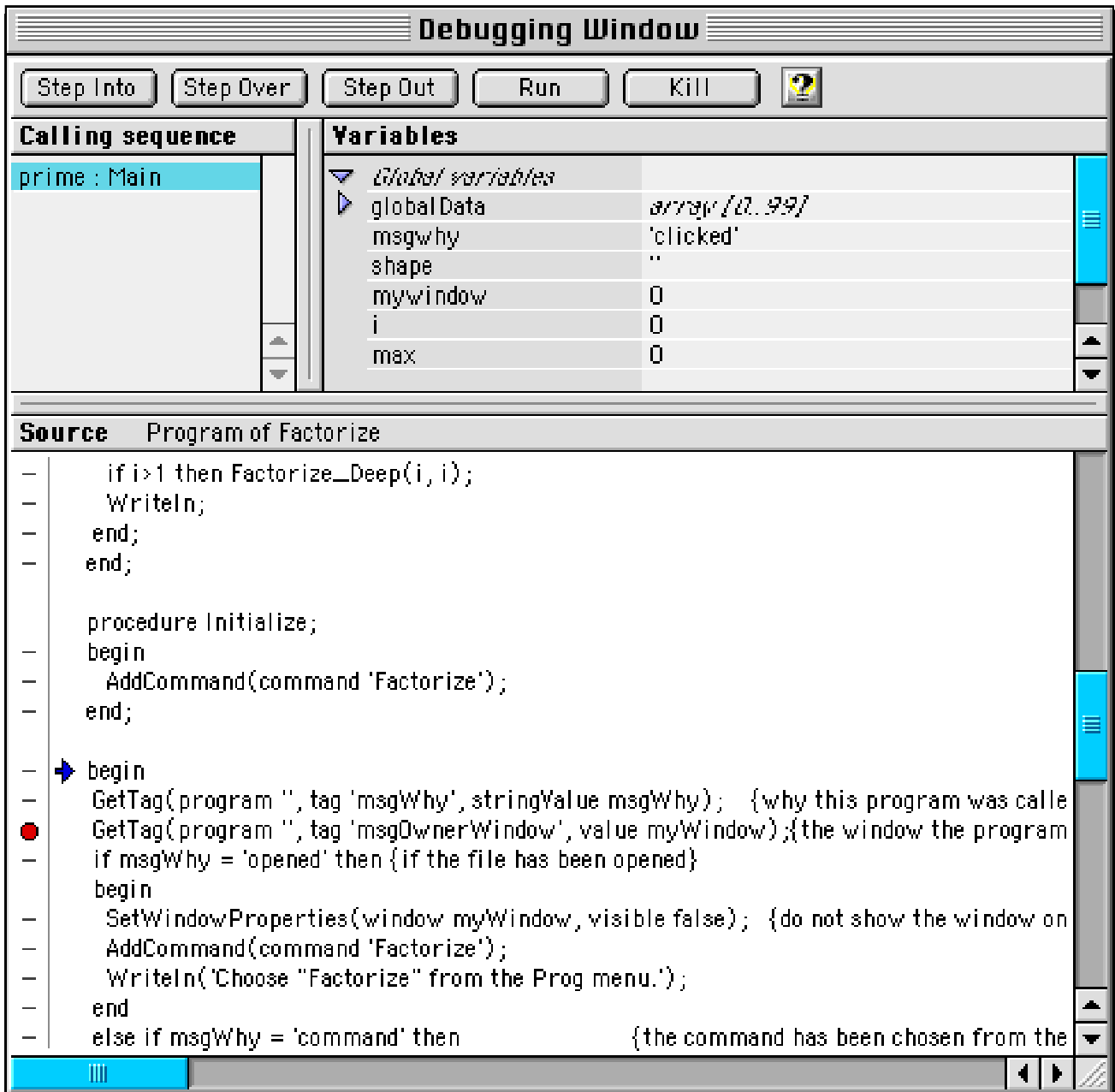
The bins can be distributed linearly, logarithmically, or according to any other scaling type supported by pro Fit. For example, if you want equidistant bins, use linear scaling, if you want to have one bin over each decade, use logarithmic scaling, etc.

Debug Window

Remember the time spent pondering about the code for a pro Fit function you wrote, looking for the place where the code could possibly be wrong and not do what you expect?

These times are over. The new **debugging window** allows to follow the execution of your code step by step and watch all variables and parameter values while you are doing this.

For debugging a program or function, check the option “Debug” at the top of its window. When you run the program or function, its debug window will show up:



Now, you can step through your program, view and modify its variables, set breakpoints, etc.

Initially the program stops at the first line of code that is executed. (Note: Some parts of your program may already be called right after compilation, such as the procedure `Initialize`. In this case, the debugging window will come up right after compilation to let you debug these parts of your code.)

The debug window has four parts:

- At the very top there is a button bar. The significance of each button is explained below.
- At the top left, the “Calling sequence” is shown. This shows through what chain of procedures and functions pro Fit went in order to reach this particular point in your code. Note that you may step through more than one of your programs and/or functions, in the case they call each other.
- At the top right, the variables that are valid at this point are displayed. You can watch and modify their values. Just double-click a value to change it. Clicking the small triangles lets you view the elements of arrays and matrices.
- At the bottom, the source of the program or function is displayed, with an arrow showing the current location.

The top row of buttons lets you control how your code is executed:

- Click **Step Into** or **Step Over** for advancing one step in your code. When you click **Step Into** and the next step is a local function or a procedure, pro Fit steps into this procedure and stops at the first instruction there. If you click **Step Over** and the next step is a function or procedure, pro Fit will execute it and stop again on the next line. If the next step is not a function or procedure, Step Into and Step Over just advance by one step.
- Click **Step Out** if you are in the midst of a local function or procedure and you want pro Fit to stop when execution returns from this function or procedure, i.e. you do not want to stop again until the function or procedure is terminated.
- Click **Run** to continue operation to the next breakpoint or (if there are no more breakpoint) to the end of your code.
- Click **Kill** to abort execution of your function or program.

You can set “breakpoints” by clicking into the left margin of the source code in the debug window. Red dots mark the breakpoints. To remove a breakpoint, click it again. When you run a program or function and pro Fit encounters such a breakpoint, execution is interrupted and the debug window comes up.

More power for user-defined functions and programs

This section is dedicated to the changes to pro Fit's program and function definition language. There are many new predefined routines available to user-defined functions and programs, providing for a better program-to-program communication (Tags), for more flexibility in number crunching (complex Matrix and Vector types), for time-based calculations (date routines), and for more flexible mechanisms for integrating program-services into pro Fit (programs attached to drawing windows and the possibility of designing a dialog box using pro Fit's drawing tools.).

A short description of the new features is presented first. An alphabetical list of all routines with all details follows later.

Input and Output

Input now allows string expressions for parameter names

```
Input('$P'+item1+';' +item2+'$value', x)
```

Note, however, that the first entry in the expression *must* be a constant string.

SetCurrentWindow() now also works for any function window. Call it for redirecting the output of **Write** and **WriteLn** calls. Calling **SetCurrentWindow(0)** resets the text output to the Results window.

Date and time management

pro Fit 5.5 can handle date and time data for all its operations. Global options for formatting such data can be selected with the preferences command. In addition to this, the following routines are provided for converting dates and times to numbers and back.

<code>NumToDateTimeStr,</code> <code>DateTimeStrToNum</code>	convert data & time numbers (seconds since 1.1.1904) into data & time strings and vice versa.
<code>NumToRelTimeStr,</code> <code>RelTimeStrToNum</code>	convert relative times (seconds) into relative time strings and vice versa. A relative time can be the difference of two dates.

Mathematical operators, functions, and procedures

For your convenience and for special purposes we have added the following mathematical functions and procedures:

Operators

<code>mod, div</code>	integer modulo and integer division
-----------------------	-------------------------------------


```

program SomeMatrixAndVectorCalculations;
var    m1,m2:    matrix[2];
        mm,mm2:  matrix[3];
        v1,v2:   vector[2];
        c:       complex;
begin
    mm1:=matr3(1+ii*2,2*ii,3,4,5,6,7,8,9);
    mm2:=1/mm1;
    m1:=matr1(1,2,3,4);
    m2:=sqr(m1*4.2)+3.3;
    v1:=vect2(1,2+ii);
    v2:=m2*v1;
    c:=v1*v2;
end;

```

Mathematical operations between matrices and matrices, matrices and vectors, vectors and vectors, and matrices/vectors with numbers do the expected thing. In the table below, “m” stands for any `matrix[n]` type, “v” for any `vector[n]` type, and “c” stands for any complex or real number.

<i>operation</i>	<i>description</i>
<code>m*m</code>	matrix multiplication, result is a matrix.
<code>m*v</code>	matrix times vector, both must have the same dimension, result is a vector
<code>m*c, v*c</code>	multiplication by a scalar. Every matrix or vector element is multiplied by c.
<code>1/m</code>	this is the inverse of the matrix m. Produces a run-time error if the matrix cannot be inverted. $1/m = \text{adjoint}(m) / \text{determinant}(m)$
<code>m1/m2</code>	matrix division. m1 is multiplied with the inverse of m2.
<code>v1*v2</code>	scalar product between two vectors. The result is a number.
<code>abs(v)</code>	the absolute value of a vector. The result is a real number.
<code>sqr(v), sqr(m)</code>	translates to <code>v*v</code> , and <code>m*m</code> , respectively
<code>conj(v), conj(m)</code>	the complex conjugate is obtained by taking the complex conjugate of each individual element.

Creating and Saving Text files

Some existing routines have been modified to allow to overwrite or append text to existing files, or to handle directories.

<code>CreateTextFile</code>	Create a text file. Get and Set Directory information.
<code>GetFileDirectory</code>	Get the directory where a given file resides.
<code>SelectDirectory</code>	Select a directory from a dialog box.
<code>SetDefaultDirectory</code>	Set the default directory used to save files without a full path name.

Strings

The compiler now interprets two consecutive quotes in a string as a single quote and not as a string terminator. Example: `write('don't')` produces the output: don't

In addition to this, pro Fit 5.5 knows the following new string manipulation routines.

<code>InsertString</code>	inserts a string into another at a selectable position
<code>CopyString</code>	copies a substring from a given string

Data Processing

The Bin command in the Calc menu is also available from programs:

<code>BinData</code>	Puts data into bins, i.e. consecutive intervals. Used to prepare data for histograms
----------------------	--------------------------------------------------------------------------------------

Tags: Sending and getting data from a program.

Tags are pieces of data that can be attached to many different things, and that can be used to store data somewhere, or to pass it to other programs. They offer a more flexible way of inter-program communication than the old GlobalData of pro Fit 5.1.2

Tags can be attached and retrieved from a window, a program, or pro Fit itself. A tag is identified by its name and its value and it can either be a string or a number. Tags are primarily used for passing data to or between programs/functions, for attaching custom data to windows or to pro Fit. Tags are identified by the object they belong to (a program, function, window, or pro Fit itself) as well as by their name.

<code>GetTag, SetTag</code>	get and set individual tags.
<code>DeleteTag</code>	deletes a tag.

Example:

```
SetTag(window 'myWindow', tag 'tag 1', value 13); {saves the tag}
GetTag(window 'myWindow', tag 'tag 1', value x); {reads the tag to x}
```

From Apple script, you can use

```
get value of tag "tag 1" of window "myWindow"
```

Attaching programs to drawing windows

Programs can be attached to drawing windows. Such programs are called whenever there is a user-interaction with drawing windows, *e.g.* when they are clicked, opened, closed, etc. This feature is useful when using the drawing window to design an interface for a program. The attached program can then read the actions of a user, and interpret them.

To attach a program to a drawing window or to modify an attached program, bring the drawing window to front, choose GetInfo from the File menu and check "Show program window". Then click OK. Alternatively click into the drawing window while holding down the control key and choose "Show program window" from the contextual menu. A window with the source of the attached program appears.

Once you have defined the program, choose “Compile” from the Customize menu. The program is compiled and its code is attached to the window.

A program attached to a window (an “attached program”) communicates with pro Fit using **tags** (see below). An attached program should always check its tag `msgWhy` to find out why it was called. If this tag contains an unknown stringValue, the program should do nothing. Otherwise, it should take some action according to its needs.

The following code-snippet retrieves the “msgWhy” tag:

```
var msgWhy:String;
...
GetTag(program '', tag 'msgWhy', stringValue msgWhy);
```

The tag `msgWhy` can currently have the following stringValues:

- | | |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'clicked': | The drawing window was clicked. In this case the tag “msgShape” will have a stringValue set to the name of the clicked shape (if a shape was clicked) or will have an empty stringValue if no shape was clicked. The tags 'msgClickedX' and 'msgClickedY' contain the clicked coordinates. |
| 'control clicked': | A control shape was clicked successfully. In this case, the tag 'msgShape' has a stringValue set to the name of the clicked shape, The tags 'msgClickedX' and 'msgClickedY' contain the clicked coordinates. |
| 'control keydown start': | A control receives keyboard input. This tag message is sent before the key is processed. In this case, the tag 'msgShape' has a stringValue set to the name of the shape, The tag 'msgCharCode' has a stringValue of length 1 giving the char code of the pressed key. (For easier comparison there are the following charCodes constants predefined: <code>charHome</code> , <code>charEnter</code> , <code>charEnd</code> , <code>charBackspace</code> , <code>charTab</code> , <code>charLf</code> , <code>charPageUp</code> , <code>charPageDown</code> , <code>charCr</code> , <code>charEsc</code> , <code>charArrowLeft</code> , <code>charArrowRight</code> , <code>charArrowUp</code> , <code>charArrowDown</code> , <code>charDelete</code> . The tag 'msgKeyCode' has a stringValue of length 1 giving the key code of the pressed key. The tag 'msgModifiers' has a value set to the keyboard modifiers (it tells, e.g. if the option-key was pressed). You can change the <code>msgCharCode</code> , <code>msgKeyCode</code> and <code>msgModifiers</code> tag to change the keyboard event before it is processed. (For easier comparison there are the following modifier codes predefined: <code>modButtonState</code> , <code>modCommand</code> , <code>modShift</code> , <code>modAlphaLock</code> , <code>modOption</code> , <code>modControl</code> .) You can set <code>msgCharCode</code> to an empty string to suppress the event. |
| 'control keydown end': | A control has received keyboard input. Called after the key is processed. Same parameter as for message “control keydown start”. |
| 'opened': | The drawing window was opened. |
| 'save': | The drawing window will be saved. |

'close': The drawing window will be closed.

'command': A command added by the procedure AddCommand has been called. The tag "msgCommand" contains the name of the command.

'idle': The program is being called because the value in its property 'idleCallTime' corresponds to the present value of TickCount.

In addition to the tag "msgWhy", attached programs can always rely on the presence of the 'msgOwnerWindow': The value of this tag is the ID of the window to which the program is attached.

An attached program should therefore look like this:

```

program attached;
  var msgWhy: String;
begin
  GetTag(program '', tag 'msgWhy', stringValue msgWhy);
  if msgWhy = ... then      check here for known tags
    ...
end;

```

It is also possible to attach a program from another program:

```

AttachProgram          attach a program to a drawing window

```

Example:

```

AttachProgram(program 'program attached; ',
  program 'begin Beep; end;');

```

Installing menu commands from a program

It is possible to add menu commands from a program, so that the program is called with a specific message if the command is selected from the menu.

```

AddCommand,          add a command to the Prog menu
CompileText         add a function to the func menu

```

Plotting

The plotting routines

```

PlotData, PlotFunction      plot a data set or a function.

```

have acquired some new parameters: dash pattern, colour of the lines connecting data points, the title to be added to a graph, the labels to be used for the x- and y-axes, the text to be used in the legend, the graph style to be used, the names to be given to graph, legend, and graph-group, the plot type, the fill color, and the direction of filling.

Drawing

Creating or deleting drawing shapes

There are new, more flexible routines for creating drawing objects from within a program:

`NewShape`, `DeleteShape` generic routines for creating and deleting any type of shape in a drawing window. (Every object in a drawing window, e.g. rectangles, groups, graphs, legends, etc. is a shape.)

Working with control shapes

As explained above, drawing windows can contain “control shapes”, such as buttons or checkboxes. The following is a list of all control shapes and of the most important properties they have. These properties can be read by calling `GetShapeProperty` and modified through `SetShapeProperties`.



Buttons: These are simple objects that hilite when clicked. Properties:

`active`: Set to true if the button can be clicked. Set to false if it is grayed and cannot be clicked.

`value`: Usually 0. Set to 1 for hiliting the button.

`text`: The text that appears in the button.



Checkboxes: They automatically change their state when they are clicked. Properties:

`active`: Set to true if the checkbox can be clicked. Set to false if it is grayed and cannot be clicked.

`value`: 0 if not checked, 1 if checked.

`text`: The text that appears beside the checkbox.



Radio buttons: They are checked when they are clicked. They usually come in groups. The program that manages the radio buttons is responsible for unchecking all other radio buttons when one radio button is clicked. Properties:

`active`: Set to true if the checkbox can be clicked. Set to false if it is grayed and cannot be clicked.

`value`: 0 if not checked, 1 if checked.

`text`: The text that appears beside the radio button.



Text fields: These are shapes that contain editable text. Generally, text fields can be edited.

`active`: Set to true if the field can be edited. Set to false if it cannot be edited.

`value`: The numeric equivalent of the text appearing in the field. Use the function `Invalid` to check if the text corresponds to a value number.

`text`: The text that appears in the edit field.

Static text field

Static text fields: These are shapes that contain non-editable text. Properties: same as for text fields, except that active has no influence on the shape's editability.

Popup

Popup menus: Popup menu shape have several “values” which can be selected by choosing them from a pop-up menu. Properties:

`active`: Set to true if the pop-up can be clicked. Set to false if it cannot be clicked and is grayed.

`value`: The currently selected item in the pop-up menu. 1 is the first item, 2 the second item, etc.

`text`: The text that appears to the left of the pop-up.

`menuItems`: The menu items, separated by semicolons.



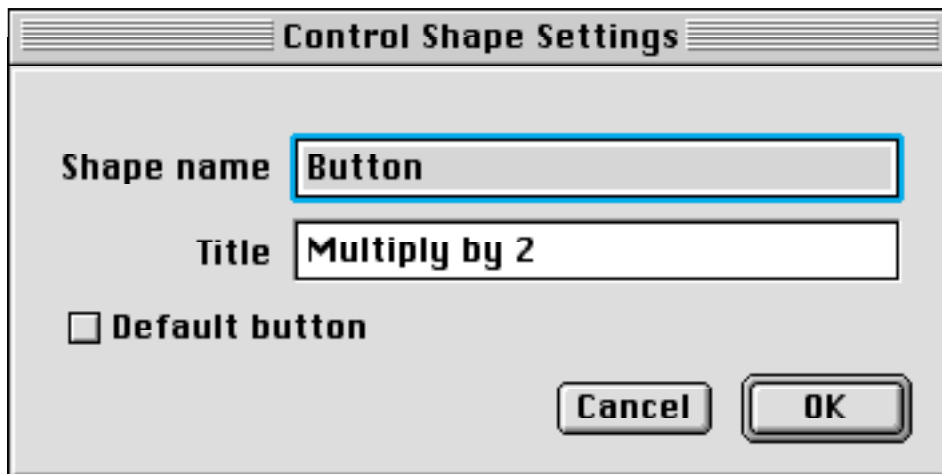
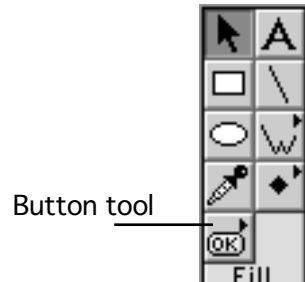
Wells: These shapes are usually used as background for other objects, e.g. a graph. They consist of a white rectangle.

For further properties that you can use for controlling these shapes, see the description of `GetShapeProperty` and `SetShapeProperties` in Appendix A.

To use control shapes, you first must draw them in a drawing window. Then you write a program that manages them and attach it to the window. Finally, you must switch the window to “dialog mode”. The following is a simple example that shows this procedure.

1. Open a drawing window and create a button named “Button”, with title “Multiply by 2”

To do this, choose the button tool from the windows toolbox. Then click into the drawing window. A dialog box appears where you can define the text that appears on the button. You can also define a name for the button. You will use the name later to access the button from a program. In this example, set the button text to “Multiply by 2” and its name to “Button”.



Click OK, and the button will appear in the drawing window.

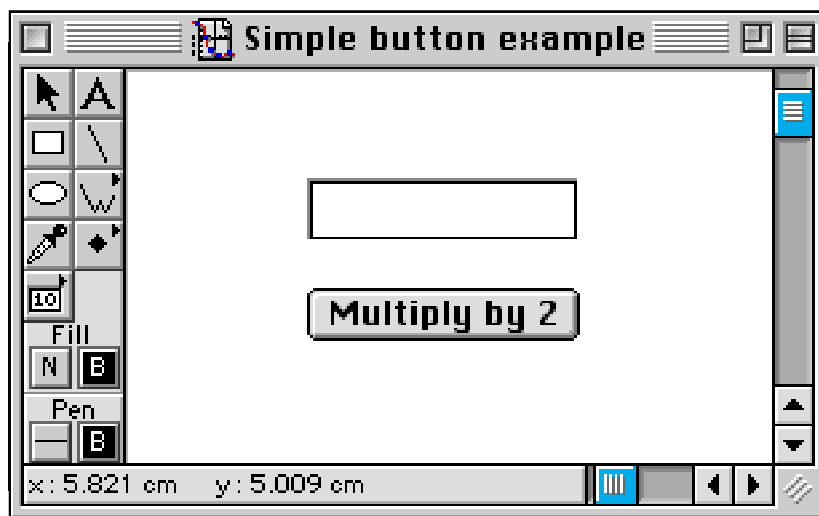
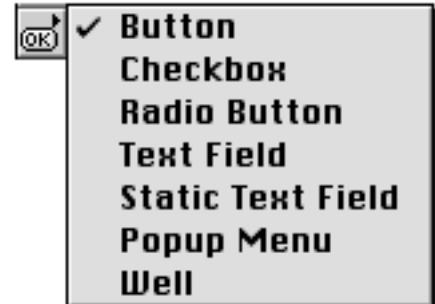


2. Create an edit field named "Number"

Now, click the button tool again and hold the mouse down until a popup menu appears. Choose "Text Field". Now, click into the drawing window and enter "Number" for the shape's name. Then click OK.

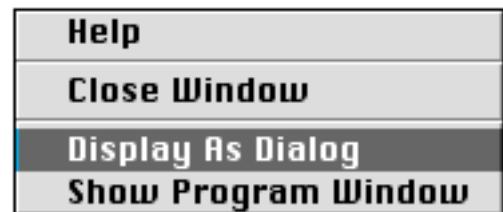
You now should have a drawing window with an edit field and a button. Arrange these items as you wish, then save the file.

The window might e.g. look like this:

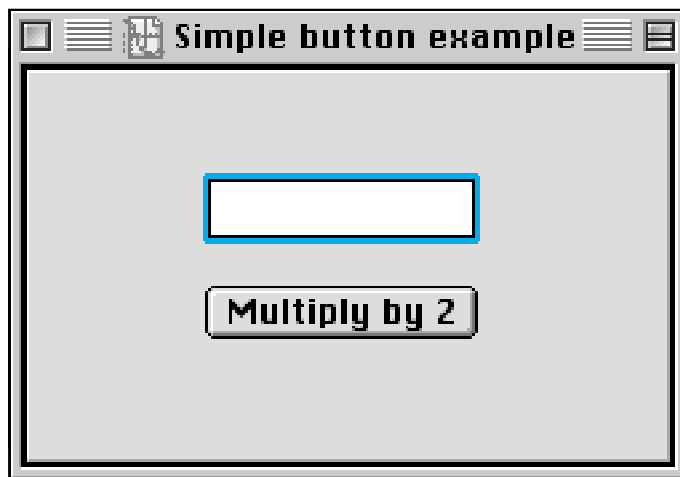


3. Switch the window to dialog mode

To do this, hold down the control key while clicking anywhere into the window and choose "Display As Dialog". Alternatively, choose "Get Info..." from the File menu and check the option "Display As Dialog".



The window now looks like a dialog box.



4. Attach the program

To attach the program, again hold down the control key while clicking anywhere into the window and choose “Show Program Window”. Then, enter the following program:

```
program attached;
  var msgWhy: String;
      msgShape: String;
      x: real;
begin
  GetTag(program '', tag 'msgWhy', stringValue msgWhy);
  if msgWhy = 'control clicked' then
  begin
    GetTag(program '', tag 'msgShape', stringValue msgShape);
    if msgShape = 'Button' then
    begin
      x := GetShapeProperty('Number', value);
      if not Invalid(x) then {if valid number}
        SetShapeProperties(shape 'Number', value x*2);
    end;
  end;
end;
```

Click the “Attach” button, or hit it Command-L, to attach the program to the window.

Now, your “dialog box” is ready to use. Enter a number in the edit field, then hit “Multiply by 2”.

Notes:

- Shape names are case-sensitive. The above program reads and writes a number from a text field named “Number”. Be careful to give it the name “Number”, and not “number”, because otherwise the program will not find it!
- If you want to modify the items in the dialog window, switch it back into drawing mode. To do this, hold down the control key and choose “Display As Drawing”. To change the text of a shape or its name, double-click it. Alternatively, select it and choose “Shape Settings...” from the Draw menu.
- As a shortcut, you can change some properties of the items when the window is still in dialog mode. To do so, hold down the command key and double-click the item you want to modify.

Getting and Setting “Properties” of various pro Fit objects

pro Fit 5.5 supports a general mechanism to retrieve or set various properties of various objects, such as the coordinates of shapes in a drawing window, the title or size of a window, etc. The following routines provide access to the properties of drawing shapes, windows, function and programs, and pro Fit itself.

Properties of drawing shapes

`SetShapeProperties,`
`GetShapeProperty` generic routines for checking, and changing any type of shape in a drawing window. (Every object in a drawing window, e.g. rectangles, groups, graphs, legends, etc, is a shape.)

Window properties

`SetWindowProperties,`
`GetWindowProperty` set or retrieve the info-text, size, title, position, etc. of a window.

Data window properties

`SetDataWindowProperties,`
`GetDataWindowProperty` get and set the number of columns and rows in the current drawing window.

Program and Function properties.

`SetFunctionProperties`
`GetFunctionProperty` get and set function and program options, hide/show function in preview window
`SetProgramProperties,`
`GetProgramProperty`

Accessing properties of the pro Fit application itself

`SetOptions, GetOption` set and retrieve various options of pro Fit.

New Pre-defined Procedures & Functions

The following is an alphabetical list of all procedures and functions which are either new or that provide new functionality in pro Fit 5.5. The individual functions and their applications have been introduced above. Here their declarations, use, and parameters are explained in detail.

AddCommand

```
procedure AddCommand(optional parameter list);
```

Adds a command to the Prog menu. Parameters:

- | | |
|----------------|-----------------------------------------------------------------------------------------------------------------|
| command | (string) The name of the command as it will appear in the menu. |
| program | (string) The program the command belongs to. Omit if the command is to belong to the currently running program. |

When the user chooses the new command in the Prog menu, the program the command belongs to is called. Before calling the program, its tag 'msgWhy' is set to 'command' and its tag 'msgCommand' set to the name of the command.

To remove a command added in this way, call `DeleteProgram`.

When the program the command belongs to is removed from the menu, the command is deleted as well.

Adjoint

```
function Adjoint(m:matrix):matrix;
```

Returns the adjoint matrix of the matrix `m`.

AttachProgram

```
procedure AttachProgram(optional parameter list);
```

Attaches a program to a drawing window. Parameters:

- | | |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| program | (string):The program (source) to attach. To add a long string, you can repeat this parameter. You can repeat this parameter if you have a long program. |
| window | (string or integer) The window's name or reference ID. Omit for front window. |

Example:

```
AttachProgram(program 'program attached; ',  
              program 'begin Beep; end;');
```

BinData

```
procedure BinData(optional parameter list);
```

Bins a given data set. "Binning" is the process of putting data into bins, i.e. consecutive intervals. For each data value, the bin or interval it belongs to is determined and the size of the bin is increased by 1. In other words, the number of values in each bin are counted. Parameters:

- | | |
|-------------------------|-----------------------------------------------------------------------------------------------------|
| window | (integer or string) The name or window ID of the window of the input data. Default: Front window |
| column | (integer) The input column (omit for using the currently selected data). |
| selectedRowsOnly | (Boolean) Set to true if you only want to bin the selected rows in the given column. Default: false |
| from | (real) The start of the bins, i.e. the left border of the first bin. |

to	(real) The end of the bins, i.e. the right border of the last bin.
nrBins	(integer) The number of bins.
scaling	(<code>linScaling</code> , <code>logScaling</code> , <code>recScaling</code> , <code>probScaling</code>): the scaling used for distributing the bins. Default: <code>linScaling</code> .
outputWindow	(name or windowID) The window to hold the resulting data. If no window with the given name exists, a new window with this name is generated. Omit for creating a new window with default name.
binCenterColumn	(integer) The column for storing the centers of the bins. Set to 0 if you don't want this column. Default: 1.
binSizeColumn	(integer) The column for storing the sizes of the bins. Set to 0 if you don't want this column. Default: 2

BitAnd

```
function BitAnd(i1,i2:integer):integer;
```

BitAnd applies the bitwise 'and'-operation on two integer numbers and returns the result as an integer number. Note that the arguments are rounded to (long) integers first.

BitClr

```
procedure BitClr(i,offset:integer);
```

BitClr sets the bit at this offset in an integer number to '0'. Note that the arguments are rounded to (long) integers first.

BitNot

```
function BitNot(i:integer):integer;
```

BitNot applies the bitwise 'not'-operation on an integer number and returns the result as an integer number. Note that the argument is rounded to a (long) integer first.

BitOr

```
function BitOr(i1,i2:integer):integer;
```

BitOr applies the bitwise 'or'-operation on two integer numbers and returns the result as an integer number. Note that the arguments are rounded to (long) integers first.

BitSet

```
procedure BitSet(i,offset:integer);
```

BitSet sets the bit at this offset in an integer number to '1'. Note that the arguments are rounded to (long) integers first.

BitShift

```
function BitShift(i,offset:integer):integer;
```

BitShift shifts the bits of the integer number by offset and returns the result as an integer number. Note that the arguments are rounded to (long) integers first.

BitTst

```
function BitTst(i,offset:integer):boolean;
```

BitTst returns true if the bit at this offset in an integer number is '1'. Note that the arguments are rounded to (long) integers first.

```
function BitXOr(i1,i2:integer):integer;
```

BitXOr applies the bitwise 'Xor'-operation on two integer numbers and returns the result as an integer number. Note that the arguments are rounded to (long) integers first.

CopyString

```
function CopyString(src:string; idx:integer; len:integer):string;
```

Returns a substring of the string `src`. `idx` defines the start of the substring (1 for first character) in `src`, `len` its length in characters. If you choose bad values for `idx` or `len`, `CopyString` returns a shorter or an empty string.

Examples:

```
CopyString('there is no way', 1, 3)           returns 'the'
CopyString('this is good', 6, 200)          returns 'is good'
CopyString('cogito ergo sum', 8, 4)         returns 'ergo'
```

CompileText

```
procedure CompileText(optional parameter list);
```

Compiles a given text. Equivalent to `Compile`, but the text containing the definition of a program or function is taken directly from the parameters of `CompileText`, instead of a window.

Parameters:

text A text containing the full definition of a program or a function.

This parameter can be repeated many times when calling this function, and the pieces of text will be automatically concatenated together.

See also: `Compile`

CreateTextFile

```
function CreateTextFile(fileName:string):integer;
```

Creates a text file with the given name in the default directory and returns a reference number used to identify this file. Call `WriteToTextFile` to redirect the output from calls to `Write` / `WriteLn` / `WriteNumber` into this file and use `CloseTextFile` to close the file when you are through.

If `fileName` is of the form '?' or '?:myName', a save dialog box is displayed. The text file will be created in the selected directory.

The parameter **opt** can take the values:

0 = new file required,
1 = append if existing text file,
2 = overwrite if existing text file,
3 = overwrite any existing file

DateTimeStrToNum

```
function DateTimeStrToNum(s:string):real;
```

Converts the string `s` into a number of seconds since 1.1.1904.

Use 'BC' placed after the year for dates 'Before Christ'.

`pro Fit` uses system routines to achieve this conversion. You may select the country specific formatting either through the system control panel or in the `pro Fit` preferences.

Best results are achieved by strings like

'14.5.1345 12:55:3.40'

(14. May 1345 at 12 o'clock, 55 minutes and 3.4 seconds).

DeleteShape

procedure DeleteShape(optional parameters);

Deletes a shape . Parameters.

shape (string) The name of the shape.
window (string or integer): The name or ID of the window that contains the shape.
Omit for using the current drawing window.

See also: NewShape

DeleteTag

procedure DeleteTag(optional parameters);

Deletes a tag. Parameters:

tag (string) The name of the tag.
program (string) The name of the program if the tag belongs to a program. Pass an empty string (' ') for the currently running program. Omit if the tag does not belong to a program.
window (string or integer) The name or id of the window the tag belongs to. Omit if the tag does not belong to a window.

See also: GetTag, SetTag

Determinant

function Determinant(m:matrix):complex;

Returns the determinant of the matrix m.

Div

operator i1 Div i2

This operator calculates the integer division of i1 and i2. Note that the arguments are rounded to (long)

Even

function Even(i:integer):boolean;

Returns true if i is an even number. Note that the argument is rounded to an (long) integer.

GetDataWindowProperty

function GetDataWindowProperty(window, property):string or real;

Returns the property of a given data window.

window: The name or ID of the window.

property: The name of the property.

In addition to the properties that you can access through `GetWindowProperty` , you can retrieve the following properties:

nrRows (integer) The number of rows in the data window
nrCols (integer) The number of columns in the data window

See also: See `GetWindowProperty`.

GetFileDirectory

function GetFileDirectory(ID:integer; var s:string):boolean;

Returns the directory where the document displayed in the given window is stored.

ID is the windowID. The path-name of the directory is returned in the string s. This function returns false if the path-name had to be truncated.

GetFunctionProperty

function GetFunctionProperty(name:string; prop):string or real;

Returns a property of the function. func is the name of the function or its index in the Func menu. prop specifies the desired property. It can be any of the properties that can be used in SetFunctionProperties. In addition to this, it can be:

nrParams (integer) The number of parameters.
name (string) The function's name

See also: SetFunctionProperty

GetOption

function GetOption(option):any type;

Returns some options of pro Fit. The type of the return value depends on the desired option. option can be any of the options you can set using SetOptions. In addition to this, it can be::

version (real): pro Fit's version number, e.g. 550 for 5.5.0
numFunctions (integer): The number of functions in pro Fit's Func menu

See also: SetOptions

GetProgramProperty

function GetProgramProperty(prog, prop):string or real;

Returns a property of the program having the given name (if prog is a string) or the given index in the Prog menu (if prog is a number). prop specifies the desired property. It can take any of the names of the properties you can set using SetProgramProperty. In addition to this, you can use

name (string) The name of the program

See also: SetProgramProperties

GetShapeProperty

procedure GetShapeProperty(shape, prop);

Gets the properties of a shape in the current drawing window. Parameters:

shape (string) The name of the shape.
xPosition, yPosition (real) The shape's x- and y-positions as they appear in the "Coords" window.
xSize, ySize (real) The shape's size in x- and y-direction as it appears in the "Coords" window.
text (string) The shape's title or text content (for control shapes and text shapes only).
value (real) The shape's value (for control shapes only). For text edit fields, this is the numeric equivalent of the edit field or nan255 (invalid) if the edit field does not contain a number.
active (boolean): True if a control shape is active, false if is disabled (for control shapes only).
rotAngle (real) The rotation angle of a drawing shape in radians.
arcStart (real) Applies only to the Oval-shape. The angle in radians at which the arc starts.
arcLength (real) Applies only to the Oval-shape. The length of the arc in radians.

See also: SetShapeProperties, NewShape, DeleteShape

```
procedure GetTag(optional parameters);
```

Get the value of a tag. Parameters:

tag	(string) The name of the tag.
program	(string) The name of the program if the tag belongs to a program. Pass an empty string (' ') for the currently running program. Omit if the tag does not belong to a program.
window	(string or integer) The name or id of the window the tag belongs to. Omit if the tag does not belong to a window.
value	(var parameter, real) The numeric value of the tag. Invalid if the tag does not exist.
stringValue	(var parameter, string) The string value of the tag. Empty if the tag does not exist.

To get a global tag (i.e. a tag that is attached to pro Fit itself), pass neither the program or window parameter. To get a tag of a program that is attached to a window, pass the program and the window parameter.

From apple script, you can use

```
get value of tag "tag 1" of window "myWindow"
```

See also: SetTag, DeleteTag

GetWindowProperty

```
function GetWindowProperty(window, property):string or real;
```

Returns the given window property.

window: The name or ID of the window.

property: The name of the property.

In addition to the properties that you can change through SetWindowProperties , you can retrieve the following properties:

paperRectLeft, paperRectTop, paperRectRight, paperRectButton	(integer) The bounds of the “paper” when the window is printed according to the settings chosen under Page Setup
pageRectLeft, pageRectTop, pageRectRight, pageRectButton	(integer) The bounds of the printable area when the window is printed according to the settings chosen under Page Setup

Example for retrieving the left border of a window:

```
i := GetWindowProperty('Untitled data', boundsLeft)
```

See also: See SetWindowProperties.

InsertString

```
procedure InsertString(src:string; dest:string; idx:integer);
```

Modifies the string dest by inserting the string src at the position defined by idx. The index idx is automatically limited to valid values. If the resulting string dest becomes too large, a runtime error occurs. idx can take values between 1 (start of string) and Length(dest).

Examples:

```
s = 'there'; InsertString('hi ', s, 1)
  returns s as 'hi there'
s = 'this good'; InsertString(' is', s, 5)
  returns s as 'this is good'
s = 'cogito ergo'; InsertString(' sum', s, 255)
  returns s as 'cogito ergo sum'
```

Mod

operator $i1 \text{ Mod } i2$

This operator calculates the integer modulo of $i1$ and $i2$. Note that the arguments are rounded to (long) integers first.

Matr2

```
function Matr2(m11,m12,m21,m22:complex):matrix[2];
```

A “construction function” for the `matrix[2]` type. It takes 4 complex (or real) parameters, assigns them to the elements of a matrix according to their order, and returns the matrix.

Matr3

```
function Matr3(m11,m12,m13, ... ,m31,m32,m33:complex):matrix[3];
```

A “construction function” for the `matrix[3]` type. It takes 9 complex (or real) parameters, assigns them to the elements of a matrix according to their order, and returns the matrix..

Matr4

```
function Matr4(m11,m12,m13, ... ,m42,m43,m44:complex):matrix[4];
```

A “construction function” for the `matrix[4]` type. It takes 16 complex (or real) parameters, assigns them to the elements of a matrix according to their order, and returns the matrix..

NewShape

```
procedure NewShape(optional parameter list);
```

Creates a new shape in the current drawing window. Parameters:

name	(string) The name of the shape.
shapeClass	(integer) The type of the shape. The following are the possible values: <code>textShape, rectangleShape, ovalShape, lineShape, polygonShape, pictureShape, graphShape, legendShape, groupShape, pointShape, subscriberShape, publisherShape, buttonShape, checkboxShape, radiobuttonShape, textControlShape</code>
xPosition, yPosition	(real) The shape's x- and y-positions as they appear in the “Coords” window.
xSize, ySize	(real) The shape's size in x- and y-direction as it appears in the “Coords” window.
text	(string) The shape's title or text content (for control shapes and text shapes only).
value	(real) The shape's value (for control shapes only). For text edit fields, this is the numeric equivalent of the edit field or <code>nan255</code> (invalid) if the edit field does not contain a number.

active	(boolean) True if a control shape is active, false if is disabled (for control shapes only).
rotAngle	(real) The rotation angle of a drawing shape in radians.
arcStart	(real) Applies only to the Oval-shape. The angle in radians at which the arc starts.
arcLength	(real) Applies only to the Oval-shape. The length of the arc in radians.
from	(string) Specifies the name of a shape that should be used as a “template” for creating this shape. The template shape specifies the defaults for the parameters of the new shape. Use this parameter for duplicating a shape. For example, command “NewShape(from ", name 'newone');” duplicates the selected shape and gives it the name 'newone'. “NewShape(from 'newone',xPosition 100)” duplicates the shape “newone” and puts the newly created shape at x coordinate 100.

See also: DeleteShape, SetShapeProperties, GetShapeProperty

NumToDateTimeStr

```
function NumToDateTimeStr(n:real; sh:integer; df:integer;
    secs:boolean; fstr:string):string;
```

Converts the date & time number *n* (seconds since 1.1.1904) into a string.

sh can take the values 1 to generate the date part only, 2 to generate the time part only, and 3 to generate both, date and time.

df defines the formatting of the date string and can take the following values: shortDate ('1/31/92'), abbrevDate ('Fri, Jan 31, 1992'), longDate ('Friday, January 31, 1992').

Set *secs* to true if seconds should be displayed as well.

pro Fit uses system routines to for this conversion. You may select the country specific formatting either through the system control panels or in the pro Fit preferences.

If *fstr* is not an empty string (''), which implies the default behaviour, it is used as a custom formatting string with various output options:

```
'%A': day of week name, long
'%B': month name, long
'%M': month
'%U': week of year
'%a': day of week name, short
'%b': month name, short
'%d': day
'%h': hour
'%j': day of year
'%m': minute
'%s': second
'%w': day of week
'%y': year
```

NumToRelTimeStr

```
function NumToRelTimeStr(n:real; dec:integer; form:integer):string;
```

Converts the relative time *n* (seconds) into a string.

dec is the number of digits of the seconds.

form lets you select the units to be displayed, 0 is standard; add 1 for seconds, 2 for minutes, 4 for hours, 8 for days, 16 for weeks, 32 for months, 64 for years and 128 for centuries.

An example of a relative time string may look like: '4c 24y 9M 2w 5d 23h 52m 44s'
with 'c' for centuries, 'y' for years, 'M' for months, 'w' for weeks, 'd' for days, 'h' for hours, 'm' for minutes, 's' for seconds.

Use the pro Fit preferences to adjust the number of days per year and month for this conversion.

Odd

```
function Odd(i:integer):boolean;
```

Returns true if *i* is an odd number. Note that the argument is rounded to an (long) integer.

Outer

```
function Outer(v1,v2:vector):matrix;
```

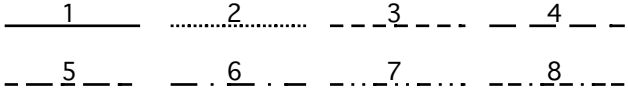
Returns the the matrix containing the outer product of its two vector parameters (of the same length *n*). Returns a *n*x*n* matrix for two vectors of length *n*.

PlotData

```
procedure PlotData(optional parameter list);
```

Plots a data set into a graph. Parameters:

xColumn, yColumn	(integer) The x- and y-columns.
window	(integer or string) The window to take the data from. You can either pass a window ID or the window's name.
autoX, autoY	(boolean) True if the limits of the x-axis (y-axis) of the graph are to be selected automatically to contain all data points, false if explicit limits are given in parameters xFirst, xLast (yFirst, yLast) . Default is false.
xFirst, xLast	(real) The start and end of the x-axis. Specify these values if you set autoX to false.
yFirst, yLast	(real) The start and end of the y-axis. Specify these values if you set autoY to false.
xScaling, yScaling	(integer) The scaling of the x- and y-axes. Values can be <code>linScaling</code> , <code>logScaling</code> , <code>recScalaing</code> (for 1/x-scaling) , <code>probScaling</code> (probability scaling) . Omit this parameter(s) to use the current default scaling.
xAxis, yAxis	(integer) The axis to be used as x- and y-axis. Omit these parameters to use the default axes.
newWindow	(boolean) True if graph is to appear in a new window, false if it is to appear in the current drawing window.
newGraph	(boolean) True if plot is to appear in a new graph, false if it is to appear in the current graph.
selRowsOnly	(boolean) True if only the currently selected rows are to be plotted, false if all rows are to be plotted.
drawErrors	(boolean) True if error bars are to be plotted, false otherwise.
pointType	(integer) Index of point type in the point style menu.
pointSize	(real) Size of point, between 2 and 128.
pointThickness	(real) Thickness of lines for drawing points: 0 (auto) , 0.25, 0.5, 1.0.
bgPointType	(integer) The same as pointType but for the background part of the point.
bgPointSize	(real) The same as pointType but for the background part of the point.
connected	(boolean) True if the data points are to be connected by lines, false otherwise. Default: false.

curveThickness	(real) Thickness of the lines connecting the data points. Omit for using default thickness.
dashPattern:	(string) Defines the length of the individual dashes, separated by commas. For example: '10,1,3,3'
curveDash	(integer) Line dash number (corresponding to the dash menu) if points are connected. Pass the position of the dash pattern in the dash popup menu. The values between 1 and 8 always correspond to: 
curveRed, curveGreen, curveBlue	(integer) The colour of the lines connecting data points. Pass values between 0 (dark) and 65535 (bright) . Omit to use the default color.
title	(string) The title to be added to a graph if you create a new graph.
xLabel, yLabel	(string) The name labels to be used for the x- and y-axes if you create a new graph.
legendText	(string) The text to be used in the legend. If this text starts with the string '\$hide\$', the curve will not appear in the legend.
style	(string) The name of the style to be used.
graphName, legendName, groupName	(string) The names to be given to the graph, legend and group shapes.
plotType	(integer) The type of the plot. Use the constants scatterPlot, verticalBarChart, horizontalBarChart, verticalSkyline, horizontalSkyline. Default: scatterPlot.
fillRed, fillGreen, fillBlue	(integer) The fill color. Pass values between 0 (dark) and 65535 (bright) . Omit to use the default color.
fillAxisX, fillAxisY	The axis to be used for filling. Omit or set to 0 to suppress filling.

PlotFunction

procedure PlotFunction(*optional parameter list*);

Plots a function into a graph. Parameters:

function	(string) The name of the function to plot. Default: current function.
xFirst, xLast	(real) The start and end of the x-axis.
yFirst, yLast	(real) The start and end of the y-axis. Specify these values if you set autoY to false.
autoY	(boolean) True if the limits of the y-axis of the graph are to be selected automatically to contain the whole plot, false if explicit limits are given in parameters yFirst, yLast. Default is false.
from, to	(real) The x-coordinates where the plot begins/ends. Default: equal to xFirst, xLast.
xScaling, yScaling	(integer) The scaling of the x- and y-axes. Values can be linScaling, logScaling, recScaling (for 1/x-scaling) , probScaling (probability scaling) . Omit this parameter(s) to use the current default scaling.
xAxis, yAxis	(integer) The axis to be used as x- and y-axis. Omit these parameters to use the default axes.

newWindow	(boolean) True if graph is to appear in a new window, false if it is to appear in the current drawing window.
newGraph	(boolean) True if plot is to appear in a new graph, false if it is to appear in the current graph.
xStep	(real) Step width for plotting. Set to 0 for using automatic step width selection.
fittedParams	(boolean) True if last fitted parameters are to be used for the function, false if the parameters in the Parameter window are to be used.
curveThickness	(real) Thickness of the curve. Omit for using default thickness.
dashPattern:	(string) Defines the length of the individual dashes, separated by commas. For example: '10,1,3,3'
curveDash	(integer) Line dash number (corresponding to the dash menu) of the curve. Pass the position of the dash pattern in the dash popup menu. The values between 1 and 8 always correspond to: <div style="text-align: center; margin: 10px 0;"> </div>
curveRed, curveGreen, curveBlue	(integer) The colour of the curve. Pass values between 0 (dark) and 65535 (bright) . Omit to use the default color.
title	(string) The title to be added to a graph if you create a new graph.
xLabel, yLabel	(string) The name labels to be used for the x- and y-axes if you create a new graph.
legendText	(string) The text to be used in the legend.
style	(string) The name of the style to be used.
graphName, legendName, groupName	(string) The names to be given to the graph, legend and group shapes.

RelTimeStrToNum

```
function RelTimeStrToNum(s:string):real;
```

Converts the relative time string *s* into a number of seconds.

An example of a relative time string may look like: '4c 24y 9M 2w 5d 23h 52m 44s' with 'c' for centuries, 'y' for years, 'M' for months, 'w' for weeks, 'd' for days, 'h' for hours, 'm' for minutes, 's' for seconds.

Use the pro Fit preferences to adjust the number of days per year and month for this conversion.

SelectDirectory

```
function SelectDirectory(s:string):boolean;
```

Displays a directory selection dialog box and returns the path-name of the selected directory in the string *s*. This function returns *false* if the path-name had to be truncated.

SetDataWindowProperties

```
procedure SetDataWindowProperties(optional parameter list);
```

Sets the properties of a data window. Parameters:

window (string or integer) The name or windowID of the window to be affected.

name	(string) The title of the window. Omit for leaving it unchanged.
boundsLeft, boundsRight, boundsTop, boundsBottom	(integer) The bounds of the window in global screen coordinates. Omit for leaving them unchanged.
info	(string) The info text attributed to the window. Omit for leaving it unchanged.
fontName	(string) The font to be used in the window. Omit for leaving it unchanged.
fontStyle	(integer) The font style to be used in the window (<code>plain</code> , <code>bold</code> , <code>italic</code> , <code>underline</code> , <code>outline</code> , <code>extended</code> , <code>condensed</code> or any sum of these values). Omit for leaving it unchanged.
fontSize	(integer) The font size to be used in the window. Omit for leaving it unchanged.
nrRows, nrCols	(integer) The number of rows/columns. Omit for leaving this unchanged.

SetDefaultDirectory

```
procedure SetDefaultDirectory(path:string);
```

Sets the default directory for saving files to the one specified by the given path-name. Pass the empty string (' ') as a parameter to re-set the default directory to its original setting, *i.e.* the directory where the pro Fit application is found.

SetFunctionProperties

```
procedure SetFunctionProperties(optional parameter list);
```

Sets the properties of a function in the Func menu. Parameters:

function	(string) The function (omit for currently selected function)
shown	(boolean) True if the function is shown in the Preview window, false if not. Omit to leave unchanged.
nrParams	(integer) The number of parameters. Omit to leave unchanged. Do not change the number of parameters while a function is being used, e.g. for fitting.

SetOptions

```
procedure SetOptions(optional parameter list);
```

Sets some options of pro Fit. Parameters:

currentFunction	(string) The current function. (Alternative call to <code>SetOptions(currentFunction 'myFunc')</code> is <code>SelectFunction('myFunc')</code>). Omit for leaving the current function unchanged.
defaultColumnType	(real): The default type for columns (<code>floatColumn</code> , <code>doubleColumn</code> , <code>textColumn</code>).
decimals	(integer) The number of decimals to be used for writing numbers in the results window. Pass a negative value for setting the total number of digits, a positive value for setting the number of digits after the decimal point. Omit for leaving the setting unchanged.
errorAlerts	(boolean) true if error alerts are to be shown when running into errors during the execution of programs or scripts. Omit for leaving this option unchanged.

scriptDebugging	(boolean) Set to true if debug info is to be printed while running AppleScripts. Omit for leaving this option unchanged.
clipboard	(string) Moves the given string onto the clipboard.
recordGraphStyles	(boolean): true if the default graph style is recorded each time a graph is modified. false to disable automatic recording of graph styles. Reset to true when finished with the execution of the program and of all other tasks, including re-drawing of windows, etc.

See also: `GetOption`

SetProgramProperties

`procedure SetProgramProperties(optional parameters);`

Sets a property of the program. Parameters are:

program	(string or integer) The name or the position in the Prog menu of the program to access. Pass an empty string (' ') for the currently running program.
idleCallTime	(integer) The next time the program will be called while pro Fit is idle. If you set this property to a non-zero value, the program will be called automatically as soon as the function TickCount returns a value that is larger or equal to idleCallTime. Before calling the program, its tag 'msgWhy' is set to the string 'idle'. The program can read this tag to determine if it has been called at idle time using the procedure <code>GetTag</code> .

SetShapeProperties

`procedure SetShapeProperties(optional parameters);`

Sets the properties of a shape in the current drawing window. Parameters:

shape	(string) The name of the shape.
xPosition, yPosition	(real) The shape's x- and y-positions as they appear in the “Coords” window.
xSize, ySize	(real) The shape's size in x- and y-direction as it appears in the “Coords” window.
text	(string) The shape's title or text content (for control shapes and text shapes only).
value	(real) The shape's value (for control shapes only). For text edit fields, this is the numeric equivalent of the edit field or nan255 (invalid) if the edit field does not contain a number.
active	(boolean): True if a control shape is active, false if is disabled (for control shapes only).
rotAngle	(real) The rotation angle of a drawing shape in radians.
arcStart	(real) Applies only to the Oval-shape. The angle in radians at which the arc starts.
arcLength	(real) Applies only to the Oval-shape. The length of the arc in radians.

See also: `GetShapeProperty`, `NewShape`, `DeleteShape`

```
procedure SetTag(optional parameters);
```

Set the value of a tag. Create the tag if it does not exist yet. Parameters:

tag	(string) The name of the tag.
program	(string) The name of the program if the tag belongs to a program. Pass an empty string (' ') for the currently running program. Omit if the tag does not belong to a program.
window	(string or integer) The name or id of the window the tag belongs to. Omit if the tag does not belong to a window.
value	(real) The numeric value of the tag.
stringValue	(string) The string value of the tag. Empty if the tag does not exist.
permanent	(boolean) True if the tag is to be saved together with the window/program (or, if the tag is a global tag, if the tag is to be saved in pro Fit's preferences files so that it is available the next time pro Fit starts up). Default: false

To set a global tag (i.e. a tag that is attached to pro Fit itself), pass neither the program or window parameter. To set a tag of a program that is attached to a window, pass the program and the window parameter.

Use either the value or the stringValue parameter, not both.

See also: GetTag, DeleteTag

SetWindowProperties

```
procedure SetWindowProperties(optional parameter list);
```

Sets the properties of a window. Parameters:

window	(string or integer) The name or windowID of the window to be affected.
name	(string) The title of the window. Omit for leaving it unchanged.
boundsLeft, boundsRight, boundsTop, boundsBottom	(integer) The bounds of the window in global screen coordinates. Omit for leaving them unchanged.
info	(string) The info text attributed to the window. Omit for leaving it unchanged.
fontName	(string) The font to be used in the window. Omit for leaving it unchanged.
fontStyle	(integer) The font style to be used in the window (plain, bold, italic, underline, outline, extended, condensed or any sum of these values) . Omit for leaving it unchanged.
fontSize	(integer) The font size to be used in the window. Omit for leaving it unchanged.
visible	(boolean) True if a window is visible.
floating	(boolean) True if the window is “floating”, false otherwise. This property can only be modified for the Preview window.
modified	(boolean) True if a window contains changes that have not yet been saved to disk.
isDialog	(boolean, drawing windows only) True if a drawing window is to be displayed in “dialog mode”, false if not.

nextShapeName (string, drawing windows only) The name of the next shape to be created. Set to blank to use a default name. Set this property before using calls such as `DrawRect` or `DrawString` for setting the name of the next shape to be generated. Ignored for non-drawing windows.

See also: `GetWindowProperty`

Transp

```
function Transp(m:matrix):matrix;
```

Returns the transposed of the matrix `m`.

Vect2

```
function Vect2(v1,v2:complex):vector[2];
```

A “construction function” for the `vector[2]` type. It takes 2 complex (or real) parameters, assigns them to the elements of a vector according to their order, and returns the vector.

Vect3

```
function Vect3(v1,v2,v3:complex):vector[3];
```

A “construction function” for the `vector[3]` type. It takes 3 complex (or real) parameters, assigns them to the elements of a vector according to their order, and returns the vector.

Vect4

```
function Vect4(v1,v2,v3,v4:complex):vector[4];
```

A “construction function” for the `vector[4]` type. It takes 4 complex (or real) parameters, assigns them to the elements of a vector according to their order, and returns the vector.