DataThief III

DataThief III manual v. 1.1

Bas Tummers

bas@datathief.org

Table of contents

1. A quick introduction

DataThief III is written in Java. This means, that apart from the "executable" called Datathief.jar, you will have to have the Java Virtual Machine. The Java Virtual Machine can be downloaded from www.java.com. Follow the instructions that are appropriate for your machine.

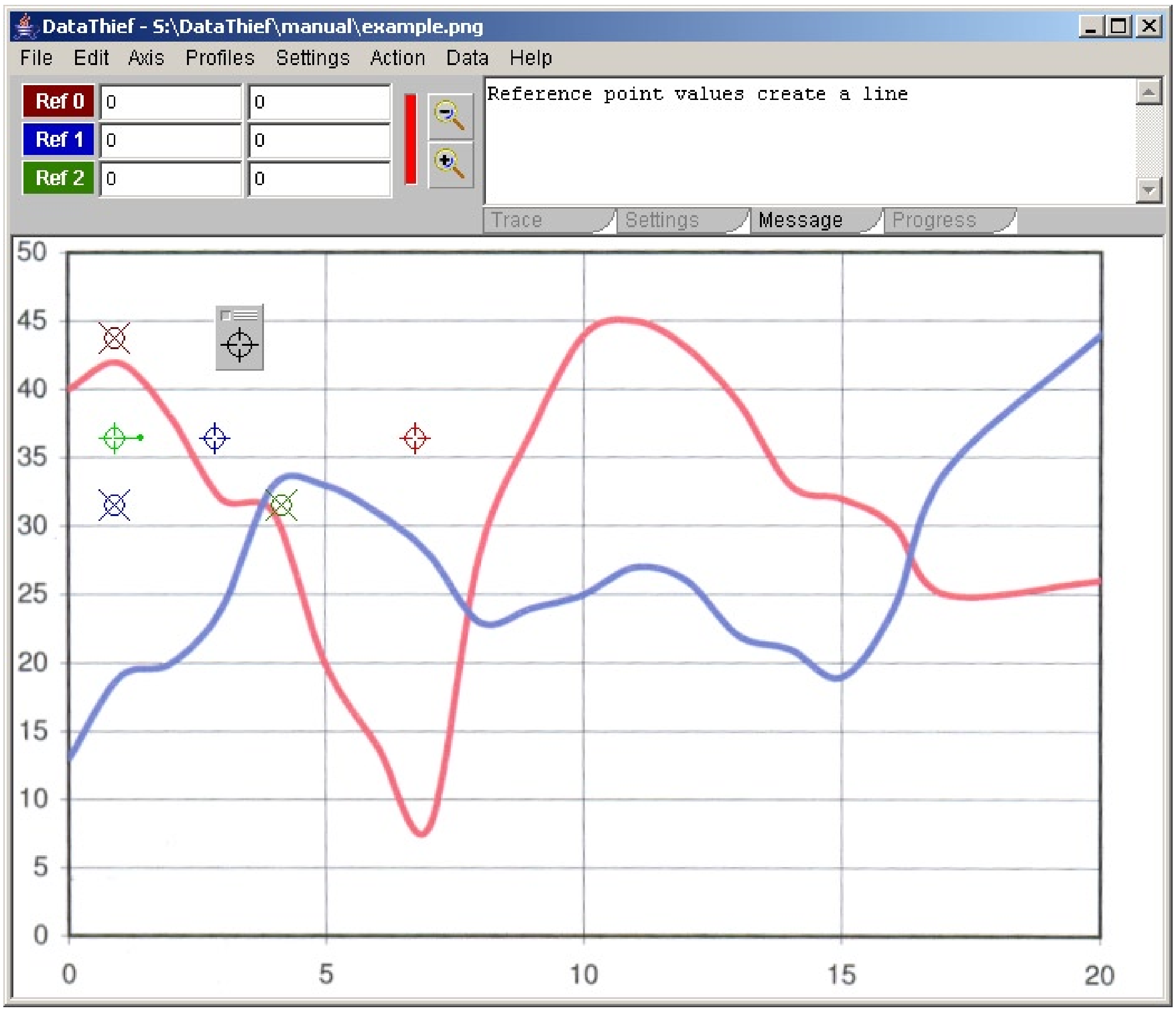
Once the virtual machine is installed, you may start DataThief.

On Windows, double click the Datathief.jar icon.

On Macintoshes with MacOS 8 or MacOS 9, double click the Datathief application icon.

On Macintoshes with MacOS X and on Linux or Unix either double click the Datathief icon, or go to the directory where you installed DataThief and type Datathief.

Once you have a running DataThief, select "Open..." from the File menu, and select the file you want to take data from. In this example, we used "example.png"



Ref 0	0	0
Ref 1	0	0
Ref 2	0	0

```
Reference point values create a line
```

On top you see the menu bar. The light gray area underneath it is the tool bar, and the white area with the image in it is the image area. In the image area you see the "Dump", that is the small gray window with the target circle in it. Also you see six location indicators. Three are coordinate indicators. These have an X through there center. The other three are the Start location indicator (green), the Stop location indicator (red) and the Color location indicator (blue). These have a + through their center.

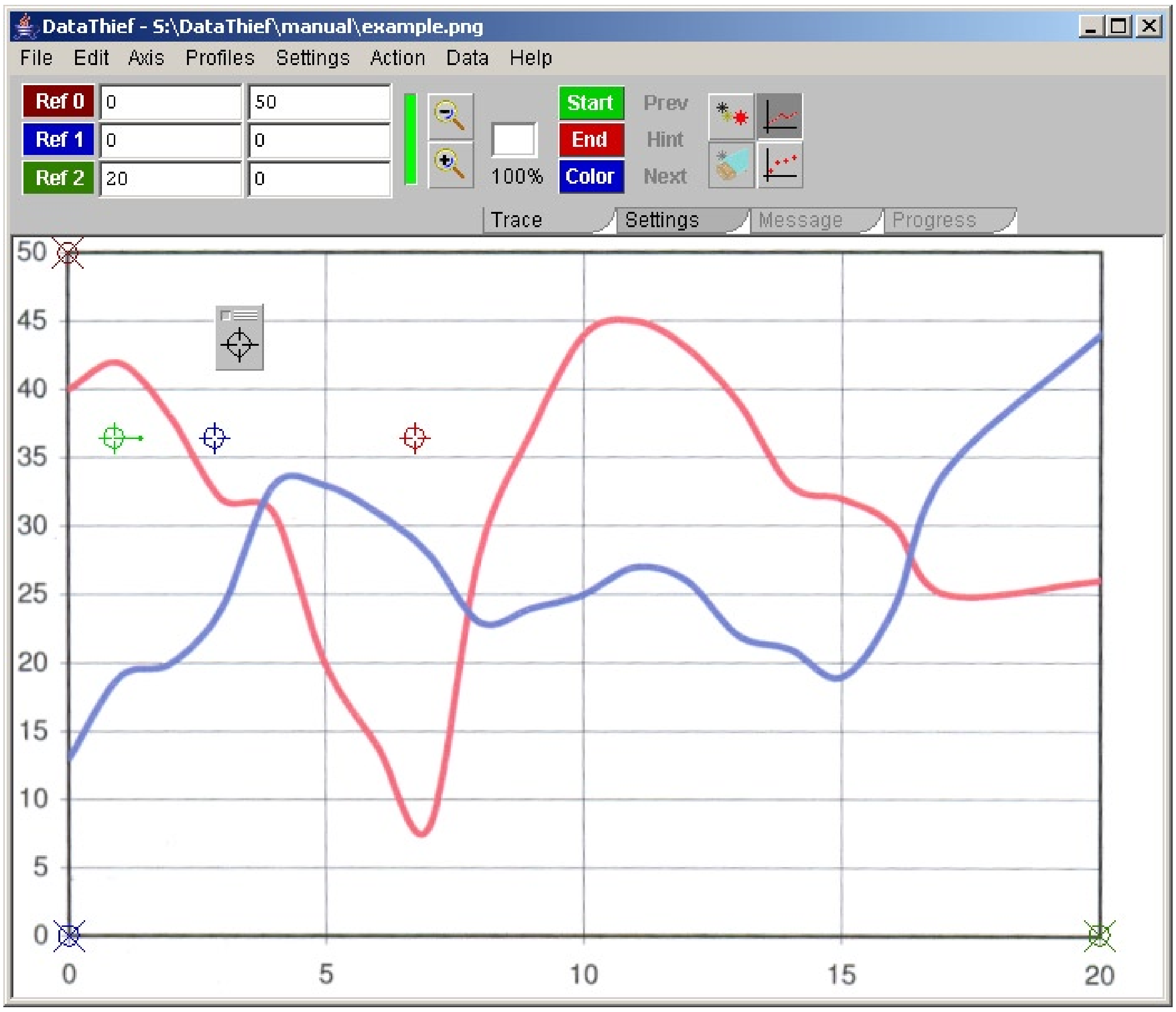
In the tool bar you see from left to right the three reference point indicators, each with it's own color. Note the difference between a location indicator, which is a circle giving a position on your graph, and a point indicator, which is a button that allows you to find the location indicator and one or two text fields that allow you to enter the coordinates for the location. After each indicator are two text fields where the coordinates of the reference points will be filled in. The shallow red vertical bar is the quality indicator. It gives an assessment of the quality of your axis definition. As we have not yet defined the axes, the quality indicator is red which means poor quality. Right from the quality indicator you see the zoom in button and zoom out button. These let you zoom in and out of the image. Right from these you see the message area stating that the "Reference point values create a line".

If you look at the values that are the coordinates for the three reference points, you will see these are all 0. So actually they even only define a point, and certainly not a valid system of axes.

The way to define your axes is:

1. Position the three reference location indicators over appropriate points. You can find the location indicator by clicking on the reference point indicator. The location indicator will flash three times through black, white and red.
2. Fill the coordinates corresponding with the location in the reference point coordinate text fields.
3. Fine tune the reference location indicators (by zooming in on them)

Once these steps are taken, our example looks like



Ref 0	0	50
Ref 1	0	0
Ref 2	20	0

Notice that the quality indicator now is fully green, showing perfect quality. Also notice that the message area is replaced by other tools.

Next to the zoom buttons you see the trace color indicator. This is the area where you can see the color you want to trace. Actually it will always display the color that is underneath the color location indicator. Underneath it is the zoom indicator, telling you the image is at it's original size. Right from these you see the Start – End – and Color point indicator. These three define which line to trace. Then you see three greyed out indicators Prev, Hint and Next. We shall cover these later. Finally you see four buttons:

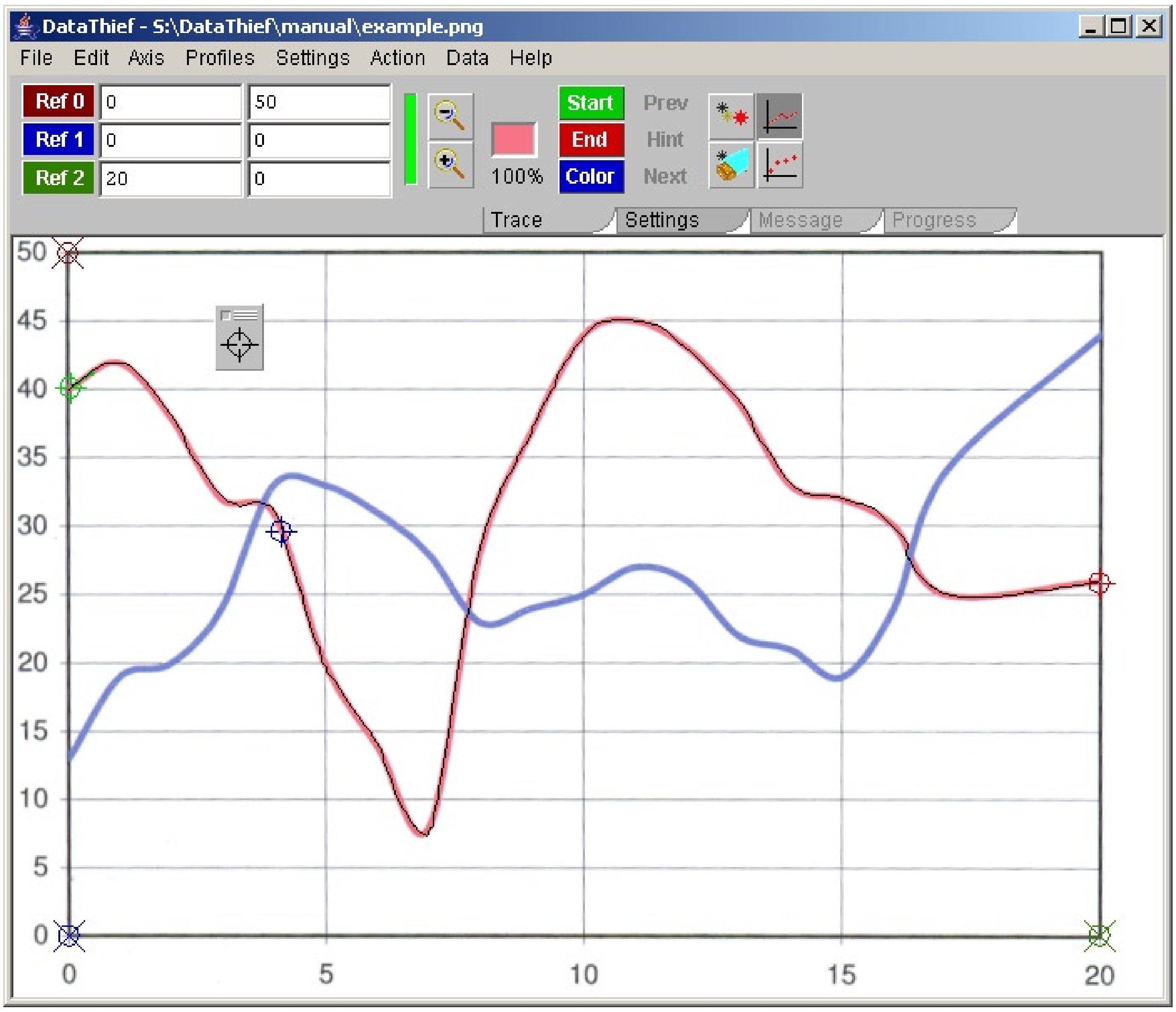
Trace Clear Trace mode Point mode

Now we shall define the line to trace. Let us follow the red line that starts at [0, 40]. The green location indicator with the small extension at the right side is the start location indicator with the extension giving DataThief an idea about the direction to start tracing (this allows you to start tracing somewhere in the middle of a line). So we shall locate it at the extreme left of the red line with the direction pointer pointing right and up. The red indicator is the end location indicator. This serves two purposes. It allows us to stop tracing in the middle of a line, but more important it prevents DataThief from retracing the path in reverse direction once the end of the line is reached. So we locate it at the right side of the red line.

The color location indicator (blue) tells DataThief which color to trace. Using this in stead of the start- or end location indicator allows you to choose the best defined location to pick the color to trace.

When you push the "Trace" button , the "Progress" will show, though on fast computers this could be gone so quickly that you will not even notice it. On slower computers it will verbalize the progress stages, and finally show the number of points traced.

Once the Trace process is finished, DataThief looks like



If you look closely, you will see that the red line has a thin black line superimposed. That is the traced line.

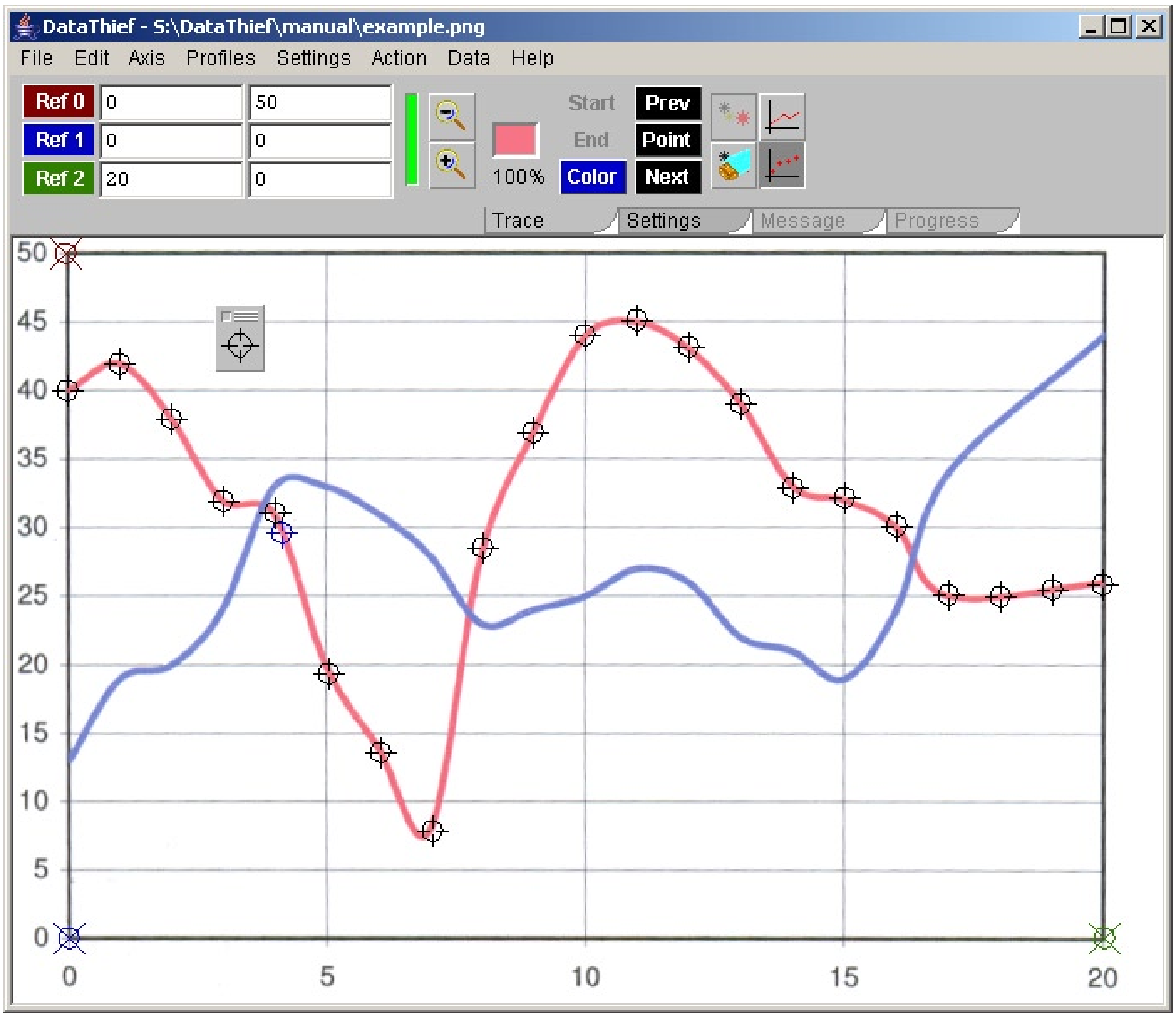
If you are interested in as much data as you can get from the image, you may now save the collected data using the "Export data.." option under the "File" menu. DataThief will offer to export the data in a file called "example.txt". In the example shown above, the file contained 990 points.

Maybe you are not interested in this amount of information, and are you happy just know the values at x being an integer value. To get this, click on the "Settings" tab in the tool panel and select "Output distance" in stead of "All points", so you may read

```
        Output distance  1       on the primary axis
```

What you ask is that The distance between points must be 1 on the primary axis, where 1 is measured in terms of the values on this axis.

Click the "Trace" tab in the tool panel, and again click the "Trace" button. You have to click the Trace button again, because the culling out process is part of the trace process, and will not be performed merely on a change of the Settings. The traced line will now follow the red path more loosely, only connecting dots that more or less coincide with whole values on the X-axis. To see these points in more detail, click the "Points mode" button


DataThief - S:\DataThief\manual\example.png
File Edit Axis Profiles Settings Action Data Help
Ref 0 0 50
Ref 1 0 0
Ref 2 20 0
100%
Start End Color
Prev Point Next
Trace Settings Message Progress
50 45 40 35 30 25 20 15 10 5 0
0 5 10 15 20

You can move the mouse pointer over any of the found points to read it's coordinates. The black buttons labelled "Prev", "Point" and "Next" may be used to browse through the points in (reversed) sequence. You can export these points, both in Trace mode and in Points mode, using "Export data" under the File menu.

Should you later want to do more with this graph, for instance trace the blue line, then it may be wise to save the settings. Select "Save" of "Save as..." From the file menu. A file dialog will appear offering to save the settings as "example.dtf". All information that is important to tracing this image is saved, including the path of the image. This means that if you move, rename or delete the image, you may still open the saved settings, but the image will not be shown.

DataThief is shareware.

If you find the program useful and intend to use it, you are requested to pay the shareware fee, see http://www.datathief.org.

The benefit from paying the shareware fee, apart from the fact that you enable the production of more shareware programs, is a key that unlocks a number of features in DataThief that will be described in this manual but are only available with this key.

These features include:

- other axes systems than the standard linear orthogonal system.
- Simple definition of systems without a clearly defined origin.
- Axes with non-numerical values, e.g. a date-based axis.
- Maintaining multiple sets of predefined settings simplifying the definition of your system.

2. DataThief full description

2.1. Loading an image

You load an image file using the "Open" menu item under the "File" menu.

The file dialog allows you to select only image files, or specific image files (like only Jpg files). Macintosh, Linux and Unix users, remember that the file selection process uses the extension of the file. So if you save your scan without an appropriate extension, the only way to see it in the open file dialog is to look at all files.

One other option the file dialog offers is to view DataThief files (dtf). Dtf files are used to save all settings from DataThief so you can work on this file later without having to define everything anew.

DataThief can read Gif, Jpg and Png files. Basically of course, the better the scan of the graph, the better the results. But there is a limit to the amount of memory that DataThief can use. If you load an image that is "too large", DataThief may issue a "Fatal error", either while loading the image or, usually, when you attempt to zoom in on the image.

2.2. Definition of Axes

2.2.1. Systems with three well defined points

The most common kind of graph we know is one with an X-axis and a Y-axis that have a clearly defined intersection point. The example.png in Chapter 1 has this intersection point at [0, 0].

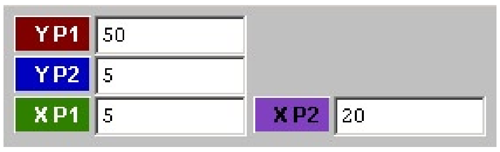
Such a system offers three points of which both the X- and the Y-coordinate is known, and these point are not in one line.

For such a system The DataThief standard mode for axes definition suffices. This means you drag – in Trace mode – the three reference location indicators to the three well defined points, fill in the appropriate coordinates at the reference point indicators and select the correct type of axis system from the Axes menu.

2.2.2. Systems without three well defined points

Suppose that in example.png from Chapter 1, there were no values given for the origin. Of course, you could assume that the crossing point of the X- and Y-axis has coordinate [0, 0], but you can not be certain of that. In such a situation, you select "4 Point ref" from the "Setting" menu.

This results in a fourth reference location indicator, and a layout of the reference point indicators that looks like



YP1	50		
YP2	5		
XP1	5	XP2	20

The layout has been made deliberately to look like a vertical Y-axis and a horizontal X-axis.

You position the two reference location indicators that correspond with YP1 and YP2 on two different points on the same vertical line. You enter the Y coordinate for those locations into the text field corresponding to the point. You do the same with XP1 and XP2 on a horizontal line. In the image above this has been done for our imaginary Example.png without it's origin values given.

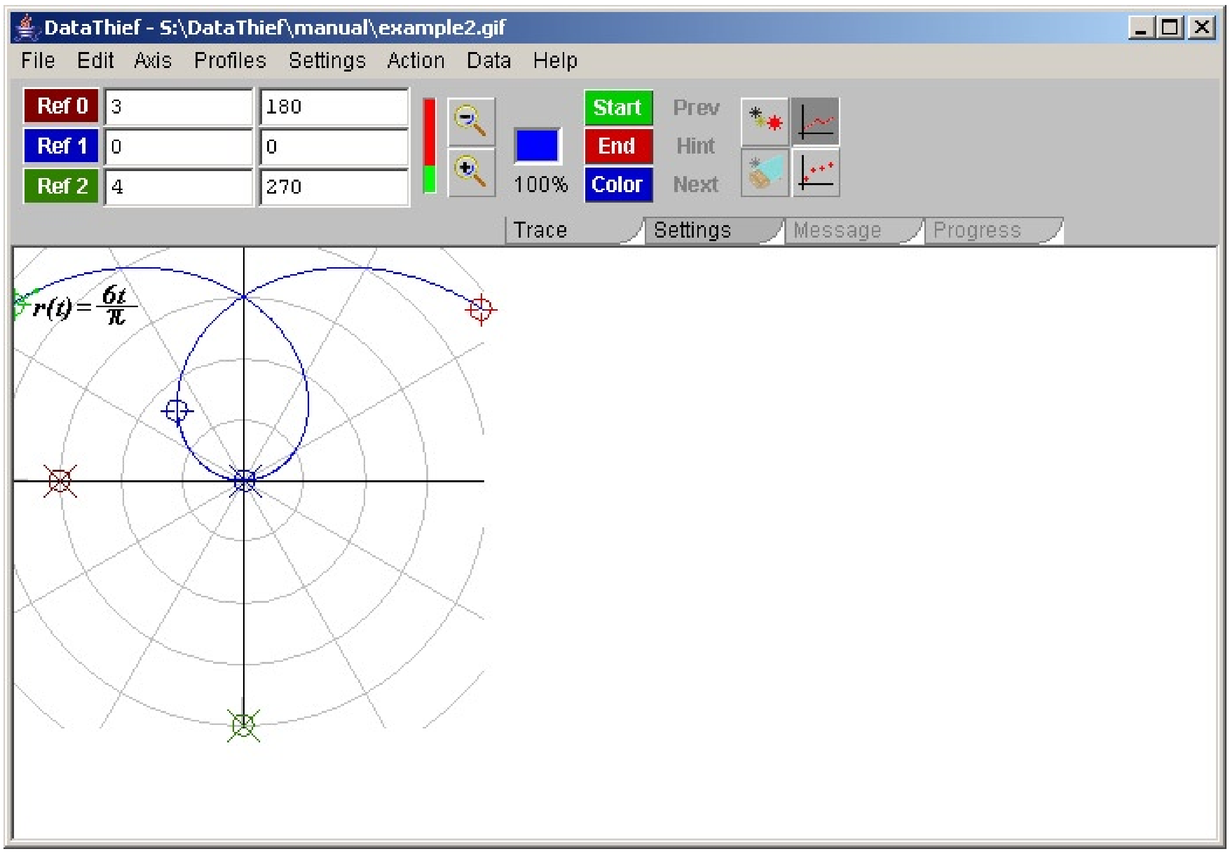
This method of defining axes will only work in systems that have independent X and Y axes, such as the standard orthogonal system. But for example in a polar coordinate system the 4 Point reference method can not work.

2.2.3. Polar coordinates

DataThief offers the possibility to trace data in polar coordinate systems or, for that matter, in any system you like to devise. But some polar coordinate systems are already predefined.

The most commonly used system is one with a linear r and φ in radians or degrees.

In the next example I have set up a system in polar coordinates, linear r, φ in degrees with 0° pointing straight to the right and φ running from 0° to 360° counter clockwise. Each circle arc stands for 1 unit in r.

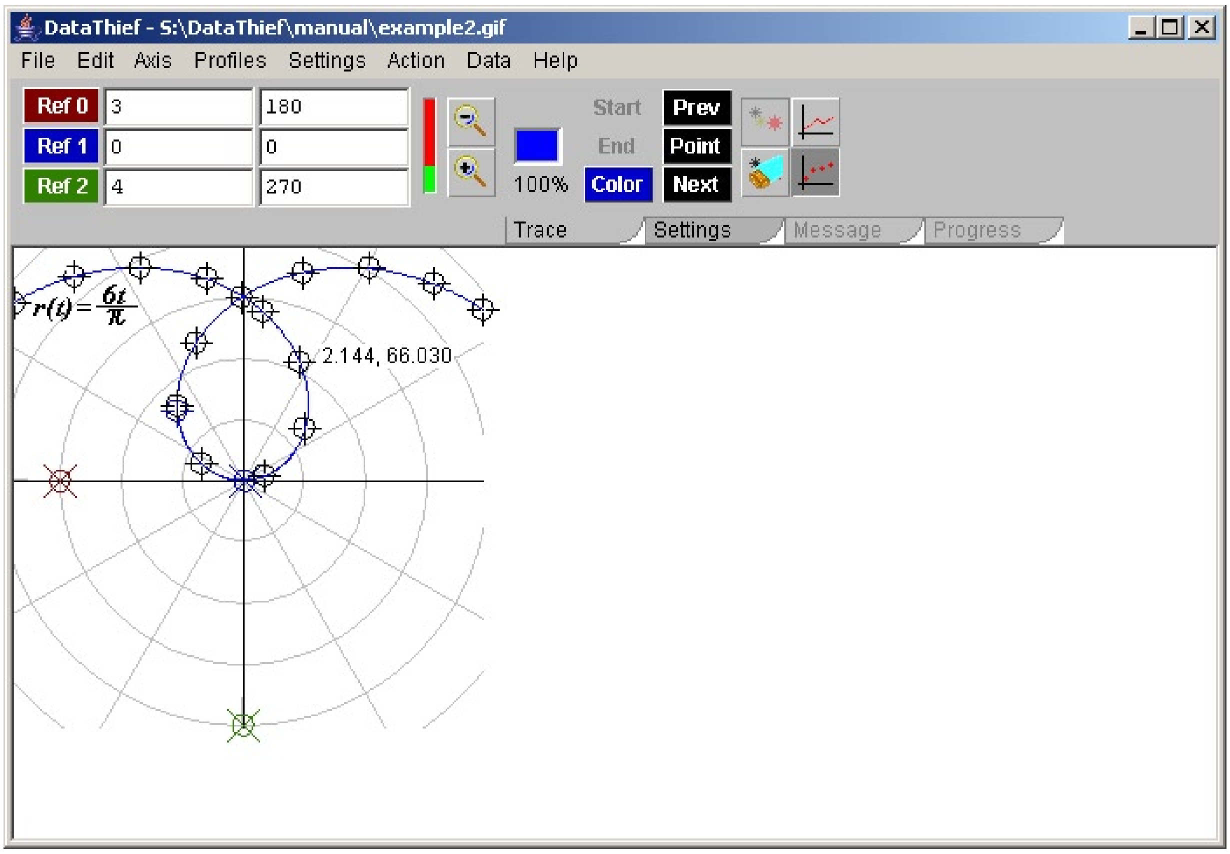

DataThief - S:\DataThief\manual\example2.gif

Ref 0	3	180
Ref 1	0	0
Ref 2	4	270

Note that the r value for Ref 0 is 3 and not -3 and of course r for Ref 2 is 4 and not -4.

Unlike earlier versions of DataThief, DataThief III is capable of tracing lines that cross themselves, usually without getting confused about where to go at the intersection point.

After setting the Start location indicator at the left most point of the blue line, the End location indication at the other end and the Color location indication on the blue line, and with a reasonable setting for which points to trace (input distance 40 on the traced path), the resulting graph in Point mode looks like


DataThief - S:\DataThief\manual\example2.gif

Ref 0	3	180
Ref 1	0	0
Ref 2	4	270

2.2.4. User defined axes

DataThief comes with a number of predefined axes, but it is impossible to predefine every axis system imaginable. Therefore it is possible to define your own system of axes. To do this, you select ,"Edit Axes" under the "Axes" menu.

But first a bit of theory behind this system.

You can imagine the image you scanned as having an orthogonal system of axes with it's origin at any location convenient for your purpose and as unit one pixel. The graph in this image has another system of axes, with a possibly other origin, axes that may have a completely different orientation, unit and distribution of units. The definition of an axis is not about defining the orientation or the units of the axis, because that is done using the reference points.

The task of describing a system of axes, is to give a translation from the image system to the graph system and back.

As a relatively simple example, let us consider a system with a logarithmic Y-axis and a linear X-axis.

Let us call the coordinates in the graph system x' and y'.

Then, the definition of the X-axis is simple:

$$x = x'$$

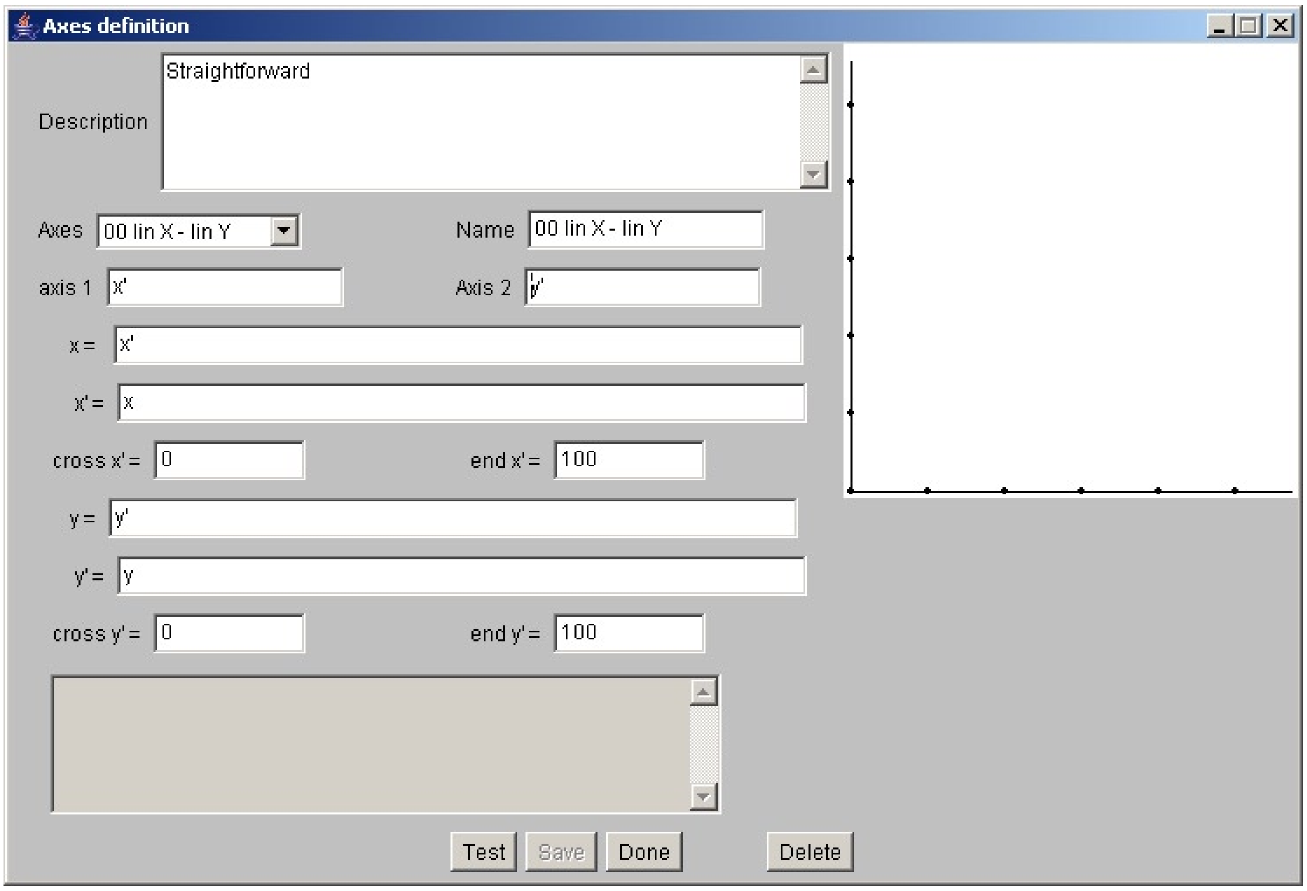
For the Y-axis it is slightly more complicated:

$$y = \ln(y')$$

The reverse translations of course read

$$x' = x \quad \text{and} \quad y' = e^y$$

Once we have these relations, we can define our axes. So, select "edit axes" from the "Axes" menu.



Description: Straightforward

Axes: 00 lin X - lin Y | Name: 00 lin X - lin Y

axis 1: x' | Axis 2: y'

x = x'

x' = x

cross x' = 0 | end x' = 100

y = y'

y' = y

cross y' = 0 | end y' = 100

You will see a dialog in which the currently active axes system is shown. Without looking at all those fields, we create a new system of axes - even though a linear X - logarithmic Y system is already present.

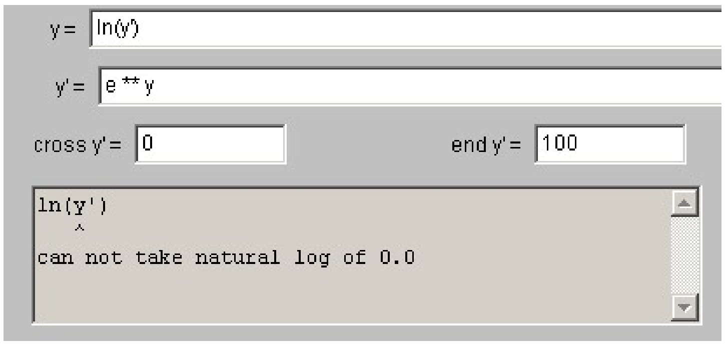
First after the word "Axes" select "*NEW*" (the lowest entry in the pick list).

All values will remain as they where, except for "Name", which is now blank. This is done to facilitate the creation of similar systems without you having to type all values again.

We use this facility now, because the X-axis remains linear. So we change the definition of the Y-axis to y = ln(y') and y' = e ** y.

Notice that "to the power" is denoted with **.

The rectangle in the upper right corner gives a simple preview of the system you have defined, but will only be refreshed when you press the "Test" button at the bottom of the window. If you should do so now, the text area nearly at the bottom of the window would display an error message

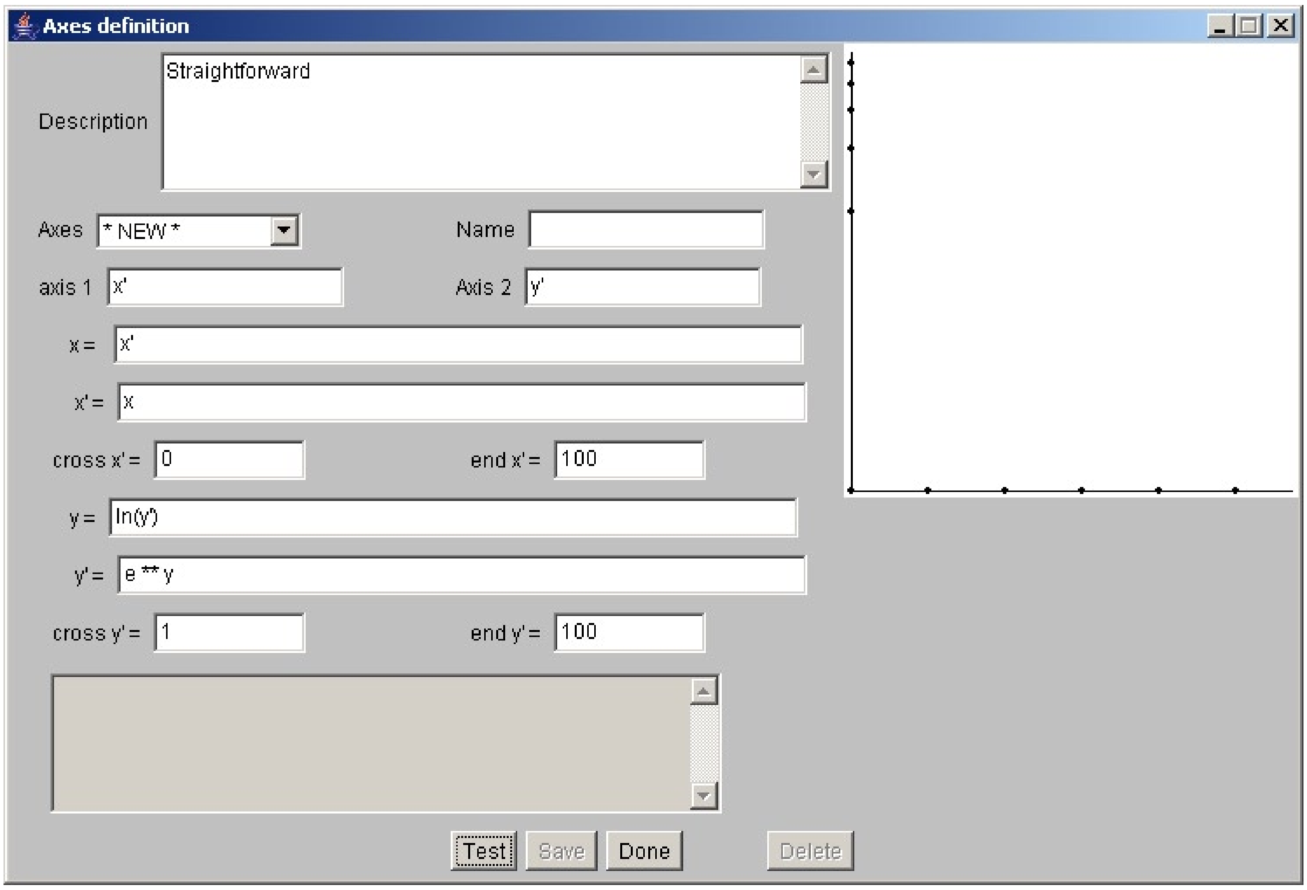

y = ln(y')

y' = e ** y

cross y' = 0 end y' = 100

```
ln(y')
   ^
can not take natural log of 0.0
```

This is because the preview area is created using a number of values between "cross y'" and "end y'". The intersection point of the two axes in the preview will be at cross x' and cross y'. But, as cross y' is still 0, This results in an attempt to take the natural logarithm of 0. Change the value of "cross y'," to 1 and press the "Test" button


Axes definition

Description: Straightforward

Axes: * NEW * Name:

axis 1: x' Axis 2: y'

x = x'

x' = x

cross x' = 0 end x' = 100

y = ln(y')

y' = e ** y

cross y' = 1 end y' = 100

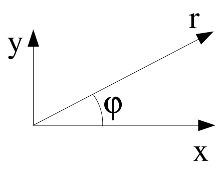
Note that the Y-axis in the preview area now has a logarithmic distribution.

You still can not press the Save button, because this system of axes has no name. You must fill in the Name field, change the Description to some sensible value, press "Test" again and then you can save this axes system.

Now let us try a more complicated system: polar coordinates.

For polar coordinates, we have to translate an x, y coordinate in the image to an r, φ coordinate in the graph.

For our convenience, we picture the origin of the image system at the origin of the graph system.



The relations are

$$x = r\sin(\varphi)$$

$$y = r\cos(\varphi)$$

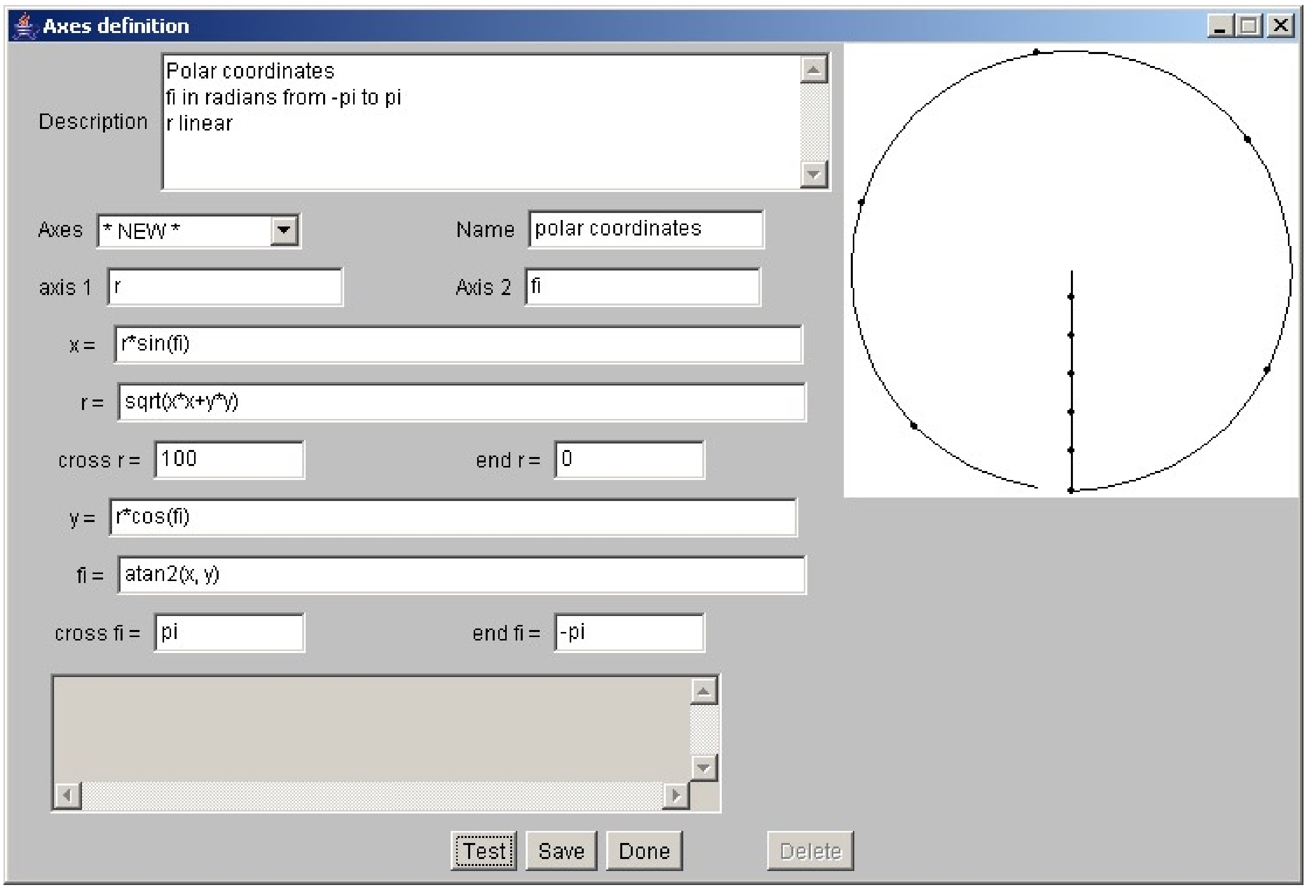
$$r = \sqrt{x^2 + y^2}$$

$$\varphi = atan\left(\frac{x}{y}\right)$$

The last relation of course only holds when y not equals 0, and the angle is in the first quadrant.

Luckily, there exists a simple function that solves this: it is called atan2, and takes the x and the y coordinate as its arguments. atan2 returns the correct angle (in radians) for every value of x and y.

So, the entire axes definition for this system looks like



Description: Polar coordinates
fi in radians from -pi to pi
r linear

Axes	* NEW *	Name	polar coordinates
axis 1	r	Axis 2	fi
x =	r*sin(fi)		
r =	sqrt(x*x+y*y)		
cross r =	100	end r =	0
y =	r*cos(fi)		
fi =	atan2(x, y)		
cross fi =	pi	end fi =	-pi

Note that I put the cross r at 100 and the end r at 0, the reverse (a cross r at 0 would result in a very small circle at best, actually it results in an error, because with r = 0 every value of φ translates to [0, 0].

Notice also that I put the names of the graph world axes in the fields behind "Axis 1" and "Axis 2".

In a linear system, you need not worry about the scale of your axis, your X-axis may run from any value to any value. In Polar coordinates that is not true. In the system described above, φ runs from -π to π. You can not change this using other values in the reference point indicators. If you want another scale for φ, you must define a new system of axes.

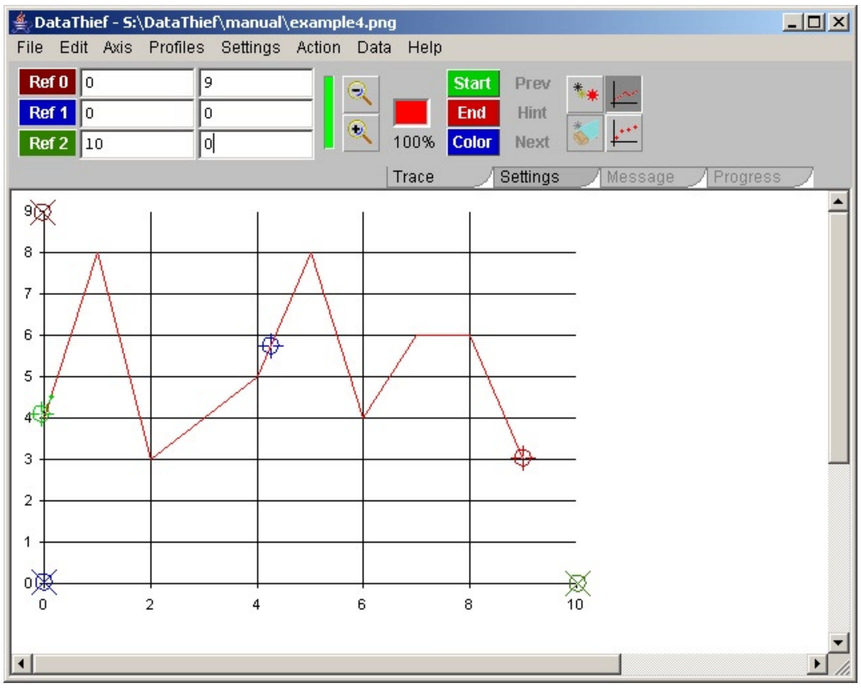
Look at the predefined axes for examples.

For a full description of available functions look at Chapter 2.9.8, DTC, DataThief Code, Predefined functions

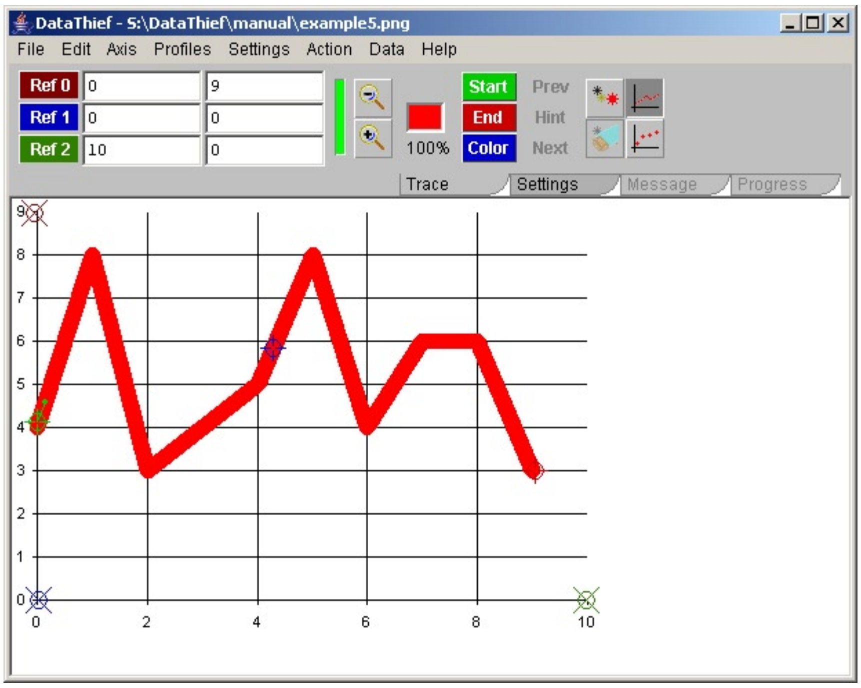
2.3. The trace process

2.3.1. Tracing

DataThief is capable of tracing quite a number of graphs. I must admit that I have also found a number of graphs that are almost untraceable. Specifically graphs with a relatively thick line and sharp corners are a source of trouble. For example


DataThief - S:\DataThief\manual\example4.png
File Edit Axis Profiles Settings Action Data Help
Ref 0 0 9
Ref 1 0 0
Ref 2 10 0
100%
Start End Color Prev Hint Next
Trace Settings Message Progress

traces without problem, whereas exactly the same data, but with a thicker line


DataThief - S:\DataThief\manual\example5.png
File Edit Axis Profiles Settings Action Data Help
Ref 0 0 9
Ref 1 0 0
Ref 2 10 0
100%
Start End Color Prev Hint Next
Trace Settings Message Progress

stops tracing at the first sharp corner (1, 8), displaying a message "Trace did not reach end point: too many doubles". Why this is, and what we can do about it is one of the subjects of this chapter.

2.3.2. Fundamentals of tracing

The basis of the trace process is of course defining what to trace. To understand this fully, I shall start with a brief description of the internals of the trace process.

The trace process is a sequence of decisions "I am at point x with direction v. What is the next point I shall visit?".

To make this decision, all points in a square with the current point at the center are evaluated. All points that do not match the requested color, and all points that where already used get a rating of 0. All other points are rated according to 3 criteria:

1. distance to the nearest point that does not match the color.
2. distance to the current point.
3. deviation from the current direction.

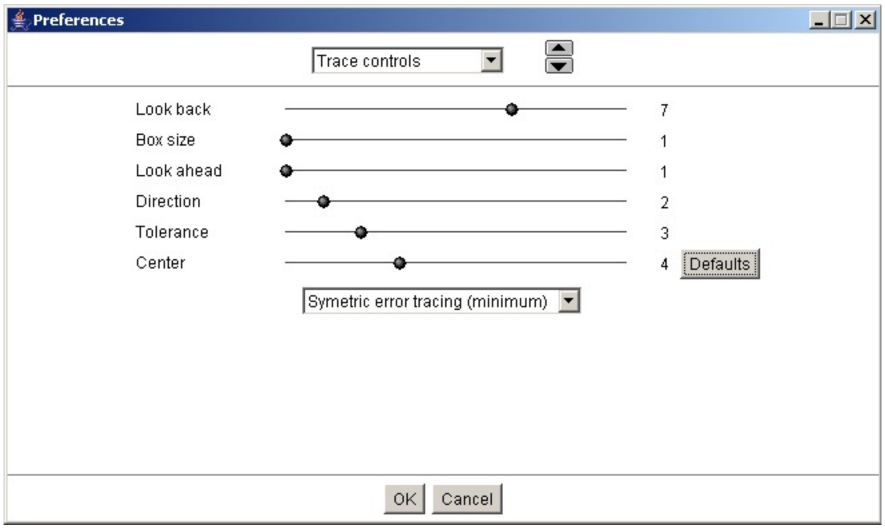
These three values result in a rating per point in the square. The best point wins. That will be the next point. The new direction is the difference between this point and a point further back. This "Further back" depends on one of the parameters you can set for the trace process.

Because "Direction" is one of the factors in the decision, this version of DataThief is capable of tracing every more or less continuous line. Relatively small gaps are no issue. Shallow lines are usually easy to trace. Sharp corners in thick lines are a problem.

So far the background of the trace process. Now to what you can do to make a successful trace.

First of course: take some care in locating the Start- End- and Color indicators. Specifically the Color indicator's location can make all the difference. The color of a scanned line is usually not uniform. There can be a lot of difference between pixels. To see precisely which pixels will take part in the trace process, select the "Settings" tab in the tool bar, and press the button "Show". The black areas in the image area take part in the trace process. Now change the "Tolerance" value by dragging the slider to the left or the right. When you release the slider, a new black/white image shows what pixels now take part in the trace process. Pushing the "Show" button once again, or selecting the "Trace" tab returns the image area to it's normal view.

There are 6 parameters that affect the trace process. You can find these under the "edit >preferences > trace controls" menu.



Look back 7
Box size 1
Look ahead 1
Direction 2
Tolerance 3
Center 4
Symetric error tracing (minimum)

Look back affects the number of points used to define the current direction. The bigger the number, the farther back the point, resulting in more reluctancy to take corners.

Box size affects the size of the square in which to look. A bigger box size allows for bigger gaps to jump, but can also result in greater computation time and a tendency to jump to other lines in the graph. The number you give here is just an indication to DataThief. Actually, the real size of the box is determined using this value, the value of "Look ahead - see following - and the thickness of the line to be scanned.

Look ahead affects the importance of nearness of the next point. The bigger the number, the less important it is that the next point is near to the current point, but DataThief will have a greater tendency to ignore intermediate points.

Direction affects the importance of maintaining the last direction (as defined by "Look back"). A bigger number will result in a reduced tendency to take turns.

Tolerance affects the deviation from the requested color that is allowed. The bigger the number, the more deviation is allowed.

Center affects the stress that is laid on being near the center of the traced line. With thin lines this has no effect, but with thicker lines an increased number will steer the traced line more through the center, sometimes overriding all other aspects, resulting in a U-turn.

Usually the default values (which you can restore using the "Defaults" button) will suffice.

The last item that affects tracing is the way error bars are traced in single point mode. We shall deal with it in the chapter concerning individual points.

All the above parameters affect the way the line is traced. There are also a number of parameters that affect the way traced points are selected. These can be found under the "Settings" tab, or under "Edit > preferences > data definition".

The selection of points can be one of

- All points
- Input distance
- Output distance

All points means that no culling is performed.

Input distance requires that you give a number that is interpreted as "pixels" and that you specify how to measure these pixels: on the primary (X) axis, or on the traced path.

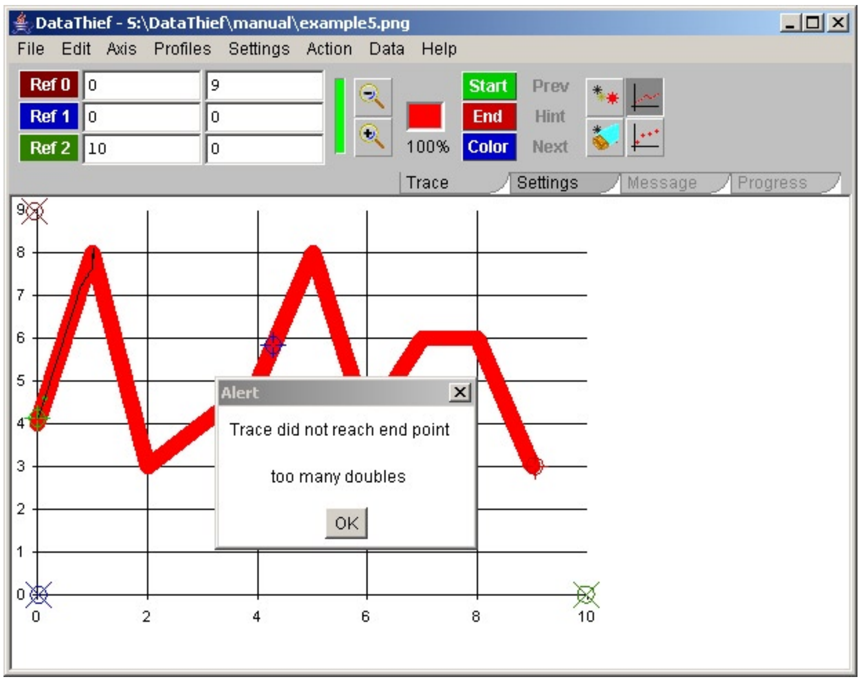
Output distance means that computations to perform culling are performed in the values of the axes. This may result in undesired results, specifically when the order of magnitude on both axes differ and you request an output distance measured on the traced path.

It is not possible to request non-linear culling, e.g. on a logarithmic x axis I would want to take a point at every whole power of 10. This can be simulated though by using Input distance.

When using Output distance on a non-numeric axis - see Chapter 2.5. Definition of output - you will have to use a translator to translate from your output format to the format used internally by DataThief. Chapter 2.5 will give an example.

2.3.3. Hints

Now we return to the examples that where shown at the beginning of this chapter. Remember the thicker line failing to trace. The message that is displayed is


DataThief - S:\DataThief\manual\example5.png
File Edit Axis Profiles Settings Action Data Help
Ref 0
Ref 1
Ref 2
Start
End
Color
Prev
Hint
Next
100%
Trace
Settings
Message
Progress

Alert

Trace did not reach end point

too many doubles

OK

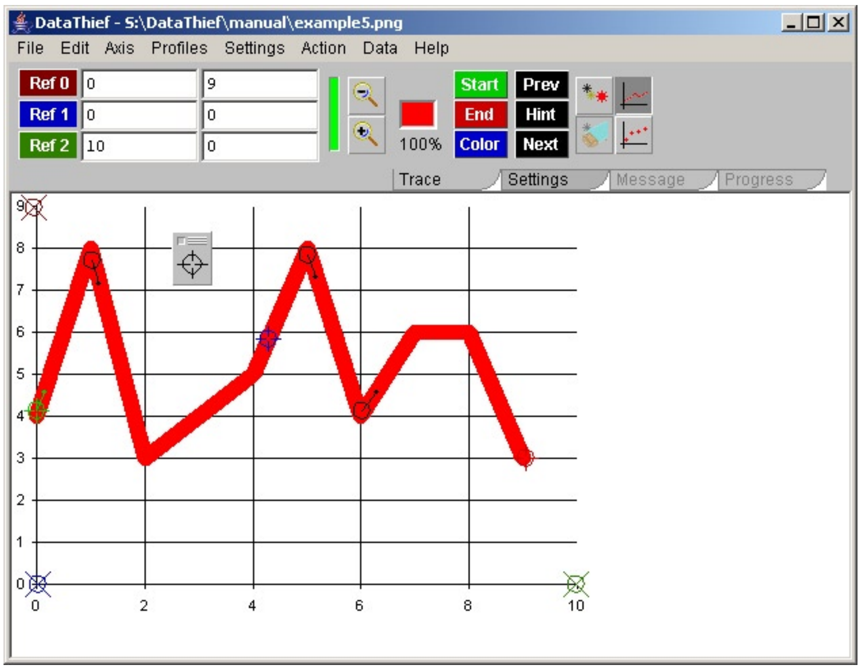
This message means that the trace process lost the way, and choose to take a U-turn. Then the process found too many points it had already visited, hence the message "Too many doubles". You can see that the problem occurred near coordinate [1, 9]. Of course we could try to modify some of the parameters, notably "Center" and "Direction" in an attempt to have the trace process succeed. But there is a simpler way.

You can see the point where the trace process lost it's way. At that point we give the process a hint about the direction to follow from here. To create a hint we need "the Dump".

The dump is the very small window with the small target in it's center. If the dump is not visible, you can make it visible by selecting "Show dump" from the "Settings" menu.

Once you have a visible dump, you can move the pointer over it. The target sign will turn red. Now you can drag a circle with a line protruding from it's right side from the dump into your scan. Locate the circle with it's center at the point where DataThief lost it's way. There you release the mouse button. When you move the mouse pointer over the end point of the line protruding from the circle you have just created, the pointer will change to a cross line. If you press the left mouse button, you can drag the end point of the line to the point where you want the trace process to resume. This is a hint. The center of the circle is the start point of the hint, and the small dot at the end of the line coming out of the circle is the end point of the hint. When the trace process come in the vicinity of the center of a hint, it is drawn towards it, and once at the center it will jump to the end of the hint, thus getting a new point and a new direction.

If you do this for the graph in this example, the trace process will fail again at [5, 8]. Create a hint at that point as well. Now the trace process will fail at [6, 4]. Creating yet another hint results in



Ref 0	0	9
Ref 1	0	0
Ref 2	10	0

And this will trace without any problem.

Notice that indeed all the problems occur at sharp bends in the curve.

Of course, the position of hints can also be of importance, so I advise to use the magnifier.

As it is cumbersome to try to scroll in a magnified image to the next hint - assuming you have more than one, you can use the buttons "Hint", "Next", and "Prev", to find a hint and move to the next or previous hint without you having to search for them.

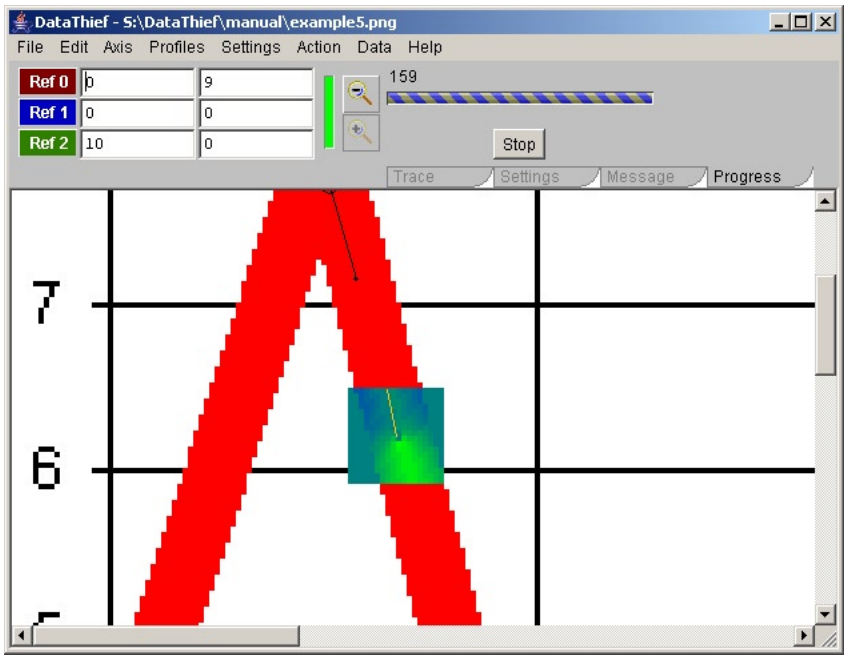
If you save your project in a dtf file, all hints are also saved.

To get a better insight in the trace process, it is possible to follow the process as it proceeds. Keep in mind, that this is a feature mainly to be used to understand why and where the trace process fails. To allow a reasonable visibility of the process, no more - though possibly much less - than 10 points per second are traced.

To use this feature, keep the shift button pressed when you start the trace process. On most systems this only works when you start the trace process using the "trace" button. Using the shift button when selecting the trace item from the Actions menu does not work

A greenish square will appear with it's center at the current point under assessment.

The following screen shot was taken about 12 seconds after the trace process was started. The fact that the progress panel shows a number of 159 points, tells us that at least some 40 points were interpolated rather than found.


DataThief - S:\DataThief\manual\example5.png
File Edit Axis Profiles Settings Action Data Help
Ref 0
Ref 1
Ref 2
159
Stop
Trace
Settings
Message
Progress

Ref 0	0	9
Ref 1	0	0
Ref 2	10	0

The lighter green areas are the preferred target points for the next position, the orange "tail" trailing the center of the rectangle shows the current direction. During this process, you can still scale the image, and follow the trace process using the scroll bars. Of course you can also still use the "Stop" button to abort the trace process.

2.4. Individual points

You can switch between Trace mode and Points mode using one of the two buttons:

Switches to Trace mode

Switches to Points mode

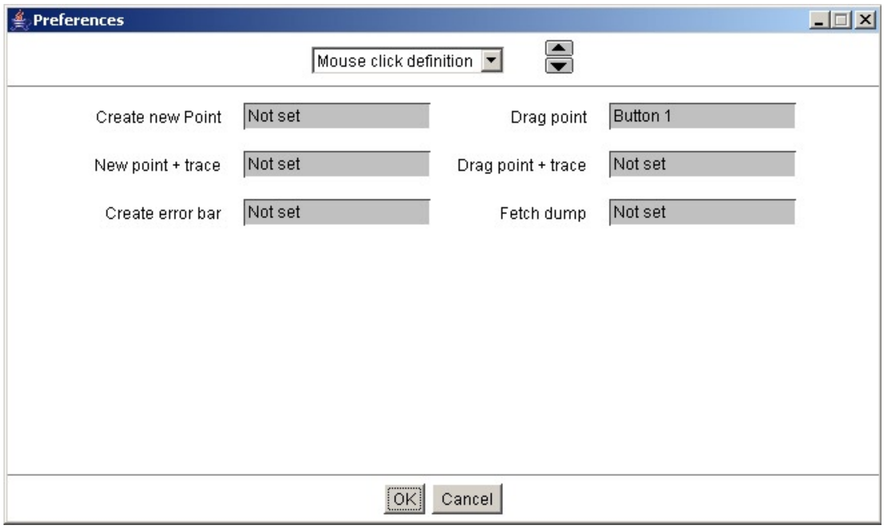
Or you can toggle between these modes using the "Points mode" item in the "Settings" menu.

If, before switching to Points mode, you had performed a trace, all data from the trace will show as individual points in Point mode.

In Points mode you can modify points that where the result of a trace, or you can create new points. Also Points mode allows you to create error bars, or even have DataThief create error bars for you.

Maybe now is the time to define your preferences in mouse clicks. By default DataThief has only one mouse click defined. That is the way to move points. But DataThief allows you to define your entirely own set of mouse actions.

This is done using the Edit > Preferences > Mouse click definition - menu item.



Create new Point	Not set	Drag point	Button 1
New point + trace	Not set	Drag point + trace	Not set
Create error bar	Not set	Fetch dump	Not set

The six grey areas you see allow you to click with whatever modifiers (Shift, Control, Alt) you like for the action.

The actions are

• Create new point	Add a point in Points mode.
• Drag point	Move a point while mouse button is pressed. Both in Points mode and in Trace mode.
• New point + trace	Create a new point and try to find it's error bars in Points mode.
• Drag point + trace	Move a point and try to adjust it's error bars in Points mode.
• Create error bar	Add an error bar to a point, or on a point with symmetric error bars, make the error bars asymmetric in Points mode.
• Fetch dump	If the Dump is hidden, make it visible and move the Dump to the mouse position, both in Point mode and in Trace mode.

Obviously, some clicks are mutually exclusive. You can not use the same mouse click to fetch the Dump and to create a new point. DataThief will solve this by setting conflicting actions to "Not set".

You may for instance set Create new point and Drag point to the same click. This will be solved by creating a new point - in Points mode - when the mouse pointer is not over an existing point. If the pointer is over a point, that point will be dragged.

DataThief does allow you to set all clicks to "Not set". This obviously makes DataThief rather useless, as there is no way you can drag the reference location indicators or the Start- Stop or Color location indicators to their correct locations.

What however even in this situation works, is dragging points from the Dump. The Dump does not look at the modifiers, and will react to any mouse click.

2.4.1. Error bars

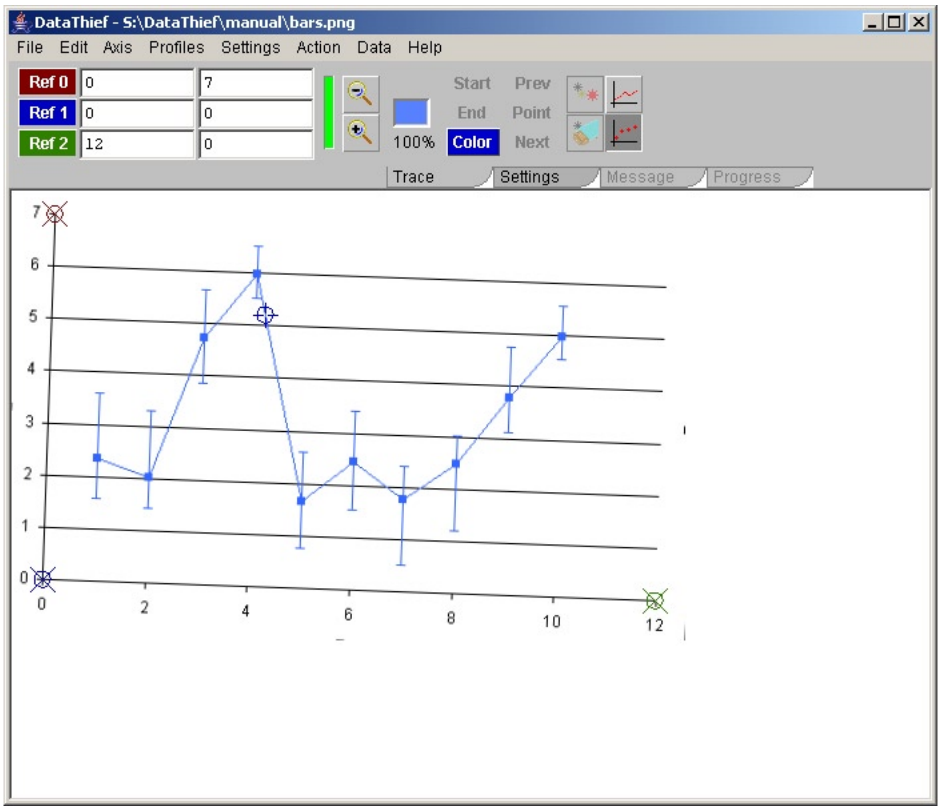
DataThief allows you to create, or trace, error bars in Points mode. Error bars are only possible in the direction of the secondary axis (the Y-axis).

Error bars come in two flavours: symmetric and asymmetric. Symmetric error bars are of the kind "value plus or minus Error" whereas asymmetric error bars read like " Value plus Error of minus Other_error".

When you move from Trace mode to Points mode, all points form trace mode are shown as points without error bars. Also, new points you drag out of the Dump or create using the mouse click action "Create new point" will be points without error bars. To change a point without error bars to one with symmetric error bars, you drag the error bar out of the point using the "Create error bar" mouse click action. It does not matter if you drag in the positive or negative Y-direction; the error bar will extend in both directions. Once the error bar is created, you may alter it using the Drag point mouse click action.

To change a symmetric error bar to an asymmetric one, drag one of the two bars using the "Create error bar" mouse click action. It is not possible to change an asymmetric error bar back to a symmetric one, nor can you change a point with error bars to one without error bars. The only way to achieve this, is to Dump the point and re-create it.

If the scanned image contains error bars, you can usually automate the process. Consider the following image (that I deliberately scanned under a slight angle).


DataThief - S:\DataThief\manual\bars.png
File Edit Axis Profiles Settings Action Data Help
Ref 0
Ref 1
Ref 2
Start
End
Prev
Point
Next
100%
Color
Trace
Settings
Message
Progress

Ref 0	0	7
Ref 1	0	0
Ref 2	12	0

This is clearly a graph with asymmetric error bars. To automatically obtain there bars, we first have to look at the one item for the "Trace settings" Preferences that we skipped earlier (see 2.3.2. Fundamentals of tracing).

Under Edit > Preferences > Trace controls you see a pop up menu allowing you to define how error tracing is performed. The possible settings are

• No error tracing	No automatic error tracing is performed,
• Symmetric error tracing (minimum)	Trace symmetric error bars, using the shortest of the positive and negative error bar as value for both.
• Symmetric error tracing (average)	Trace symmetric error bars using the average value.
• Symmetric error tracing (maximum)	Trace symmetric error bars, using the greatest of both values.
• Asymmetric error tracing	Trace asymmetric error bars.

Select the last. Place the Color location indicator somewhere on the blue line to define what color to trace. Then, with the appropriate mouse click, click at the center of any of the points in the graph with error bars. A new point with two extensions will appear. You can, keeping the mouse button down, move it, and watch the error bars follow suit. Once you have released the mouse button, you can still modify the point or it's error bars by using the "Drag point" mouse click, or move and automatically adjust the error bars by applying the "Drag point + trace" click.

When the data is exported, all points are examined. If there is one point with asymmetric error bars, all points will be treated as having asymmetric error bars, resulting is four values per point.

If there is no point with asymmetric error bars, but there is at least one point with symmetric error bars, all points will be treated as having symmetric error bars, resulting in three values per point.

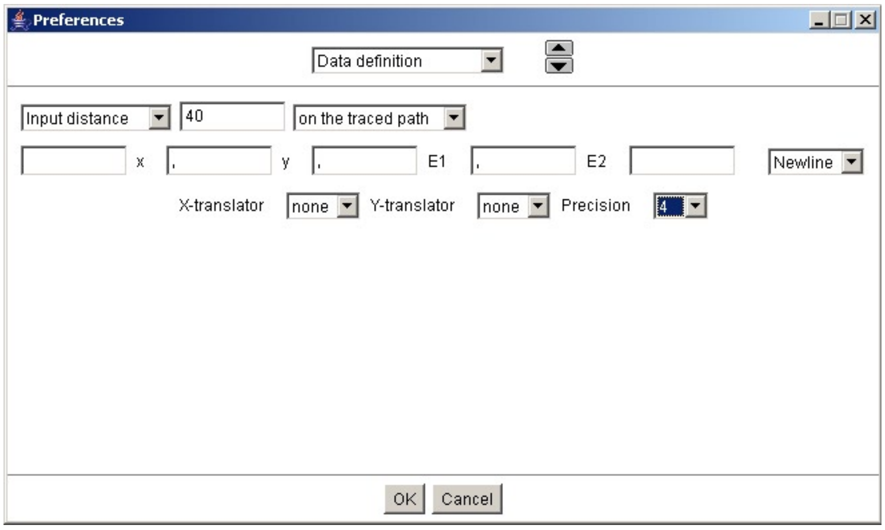
When no point has an error bar, the output will consist of two values per point.

The exact format of the output is user definable. How you can do this is one of the subjects of the next Chapter.

Error bars can also be applied to x/y dependent axes systems such as Polar coordinates. Due to the inherent computation-expensive nature of these axes systems and their translations, automatic error bar tracing will not work. Also, as in the predefined polar coordinate systems the secondary axis is φ. As the error bars are drawn as straight lines from the center of the point outward, these tend to the shape of a V in polar coordinates.

2.5. Definition of the output

The format of the file created when you export your data can to some extend be defined by you. Though some of the values, for instance the precision can be specified in the "Settings"-tab, The complete specification of your output format is done at Edit > Preferences > Data definition.



Input distance: 40, on the traced path

[ ] x [,] y [,] E1 [,] E2 [ ] Newline

X-translator: none Y-translator: none Precision: 4

Here you may specify the culling of points after a trace. This we have already seen in Chapter 2.3.2. Fundamentals of tracing.

The next line defines the exact text of your output file.

Each point consists of two, three or four values, depending on the nature of your error bars.

Regardless of your axes definition, these values are called X, Y, E1 and E2 for primary axis, secondary axis, first error value and second error value respectively.

In the line you see a each of X, Y, E1 and E2 surrounded by text fields. The rule is, that the content of the text field preceding the name is part of the output if the value corresponding with the name is present.

So, the first two text fields are always part of the output (each point has an X and a Y).

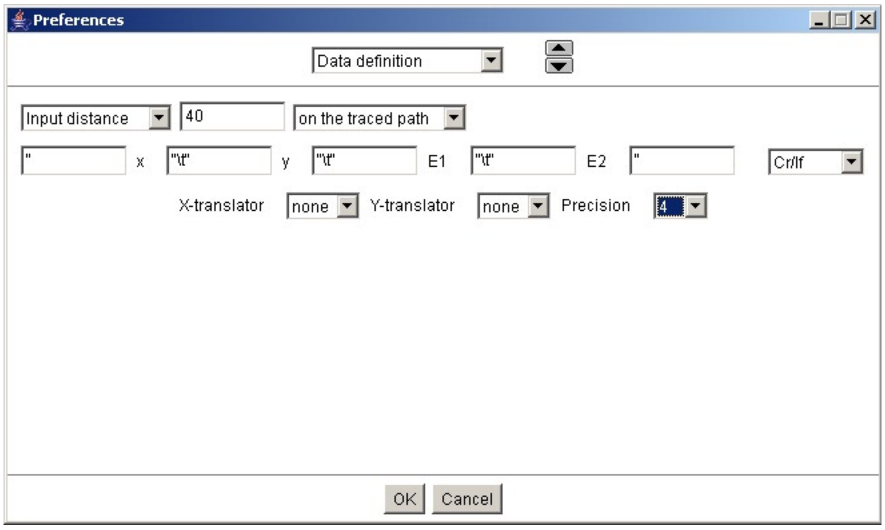
The third field is part of the output if you have error bars, and the fourth field is in the output if the error bars area asymmetric. The fifth field is always present (though by default it is empty). The last element on the line allows you to define what character you want at the end of a each point.

Some character can not be inserted in a text field. Specifically a tab will advance the focus to the next text field, whereas for some data formats a tab is essential as separator between fields. For this reason you can use the following sequences in these text fields:

sequence	inserts
\b	backspace
\s	space
\t	tab
\r	carriage return (CR)
\n	newline
\\	\
\ddd	ddd is one to three octal digits insert the ASCII value of the three octal digits. For instance to insert a formfeed (octal 012) use \012

Of course, you can create all of them with just the last, but that is not very readable. The \s is also present to increase readability, though if you prefer it, you can of course use a plain, though almost invisible, space.

So, to create tab separated values where each value is surrounded by quotes (a format that can be read by most spreadsheets), you could use



Input distance 40 on the traced path

" x "\t" y "\t" E1 "\t" E2 " Cr/lf

X-translator none Y-translator none Precision 4

Notice the quote in the fifth text field. Without error bars, the third and fourth text field are not used, so this way you will always have the closing quote.

The exported file can have a header. When you export a file, DataThief tries to execute the function

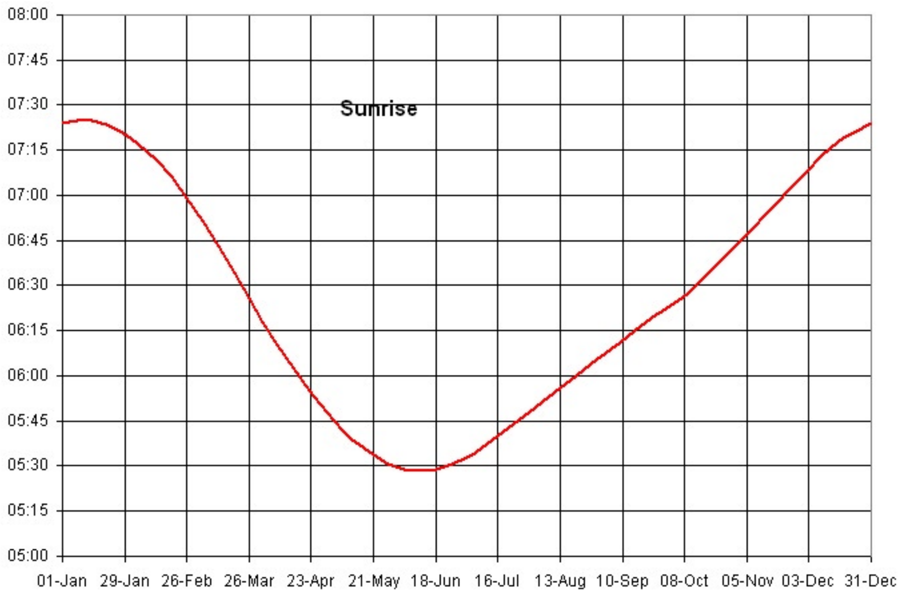
```
exportHeader()
```

If the function exists, it's result is interpreted as string and added at the top of the exported file. By default exportHeader returns a string containing

#datathief imagefile name date and time

2.5.1. Translators

Until now we have dealt with numeric axes, but what if we have a graph that looks like



We can of course do a lot of computations assigning 1 to January the first and 365 to December thirty first, but that is cumbersome and error prone.

What we would like to have is something that translates a date to a number and back, and of course for the Y-axis something to translate between a time and a number.

DataThief offers two predefined functions:

toDate	takes two arguments: a number and a string specifying the format of the date.
fromDate	takes two arguments: a string with the date and a string specifying the format.

Some of the elements that can be part of the format are

d	day of the month (one or two digits)
MMM	name of the month (three letters)
yyyy	year (four digits)

h	hour (one or two digits)
mm	minute (two digits)
ss	seconds (two digits)

A full list is given in Chapter 9.2.8.3.

With these we can make the functionality we like: for each value v on the X-axis we have

date = toDate(v, "d-MMM")

and it's reverse action is

v = toDate(date, "d-MMM")

and equally for the Y-axis

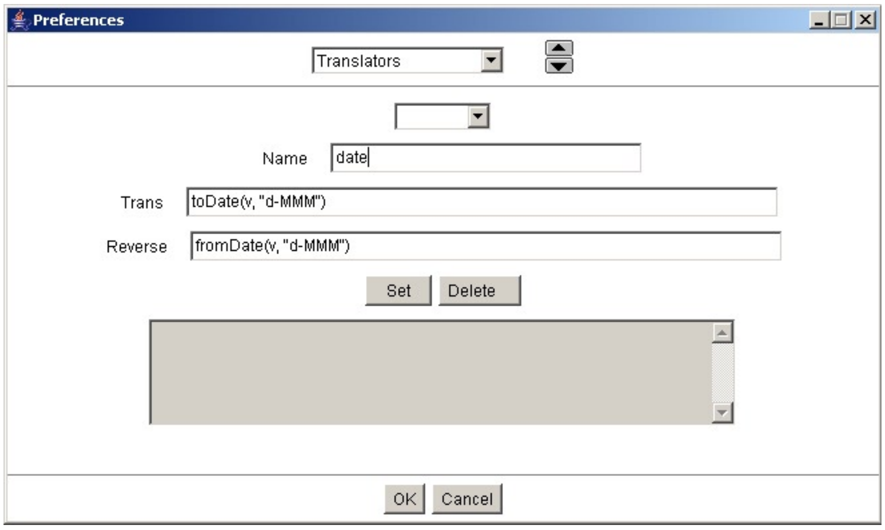
time = toDate(v, "h:mm")

and reverse

v = fromDate(time, "h;mm")

Ok, we can translate from date and time to numbers and back, but what can we do with it?

Use Edit > Preferences > Translators



Name: date

Trans: toDate(v, "d-MMM")

Reverse: fromDate(v, "d-MMM")

Here you can define translators.

A Translator is a set of two expressions that define the relation between a number and a "something". The "something" is a string. "Trans" converts the value, which is always called "v", to the "something" and "reverse" converts this "something", which is also called "v", back to the number. So the simplest Translator (called none") consists of Trans = v and Reverse = v. There is a catch though. If the result of Trans is numeric, DataThief's number formatter, using the precision you can set in the "Settings" tab is applied. If the result of Trans is a string, this will not happen. So to get unformatted numbers, you can apply a Translator with Trans = (string)v and Reverse = v.

To make it formal:

Trans gets a parameter v that is a string. The result of trans must be a number, or a string that evaluates to a number (So actually the string "2 + 3" is a valid result of Trans).

Reverse gets a float as parameter. If the result of Reverse is a string, no further formatting is applied. If the result of Reverse is an integer of float, DataThief's standard formatter (format(v, "g", -precision()) is still applied.

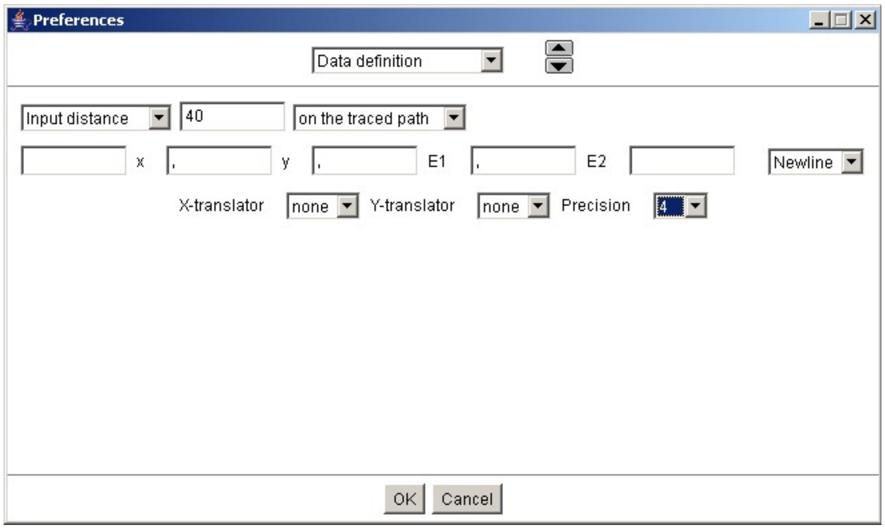
In the example above the translator for date is already entered. To add it to the list of translators press the "Set" button. The name "date" will appear in the pull-down menu. Now you can enter the definition for the time translator and again press "Set".

If you make an error in the definition of the functions, this will be reported in the gray area and the definition is not saved.

Once you have these translators defined, they are part of your DataThief environment until you decide to delete them, so you will not have to define them again.

Deleting a Translator is done by selecting the translator in the pull-down menu, pressing the "Delete" button and confirming that you want to delete the Translator.

Now return to the "Data definition" tab of preferences (select it from the pull-down menu on top of this window).



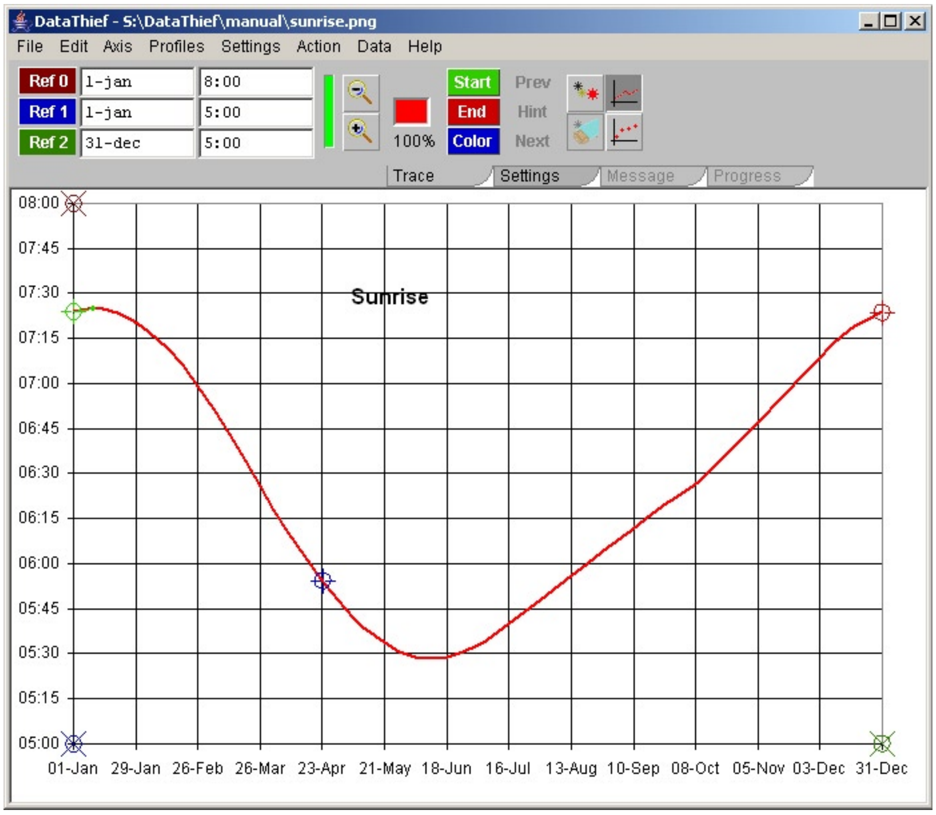
Input distance | 40 | on the traced path

x , y , E1 , E2 Newline

X-translator none Y-translator none Precision 4

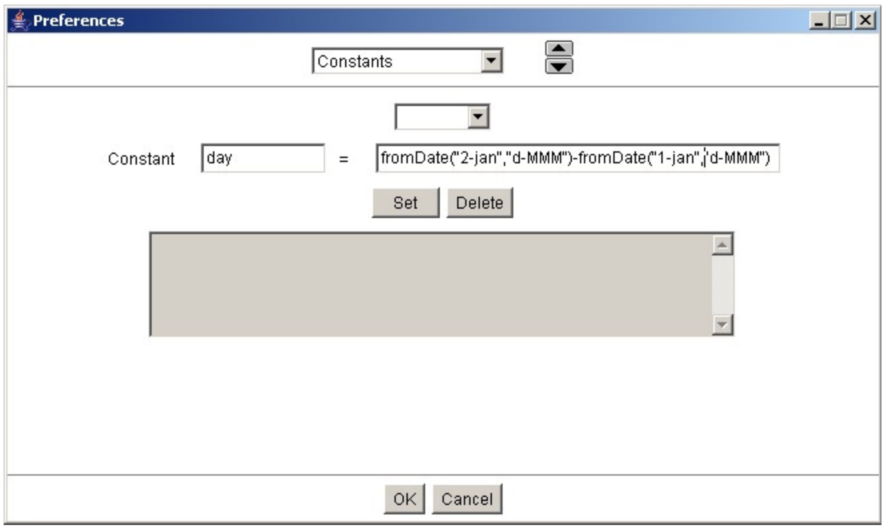
After the word x-translator select "date" (because you have defined a translator with the name date, it will appear here). After y-translator select "time".

The click "OK". It is time to define a system with non numeric data. You will probably have an error message reading "Date does not match (d-MMM)". Don't let that bother you. It really means you have not yet filled in the correct values that belong to these translators. If you have entered the correct values for your system of axes - and of course located the various locators your system should look like



Ref 0	1-jan	8:00
Ref 1	1-jan	5:00
Ref 2	31-dec	5:00

Of course you can trace this graph, but what if we would like to have a data point about every fourthnight. Problem is that we do not know the number of pixels to specify, nor do we know the number to specify as output distance (we can not just say "1-jan" nor regretfully 15-jan - 1-jan). The latter though is not far from what we can do. First, to make things easier, let us define the number corresponding to one day, and because we might want to use that number more than once, we shall make it a constant. So Edit > Preferences > Constants



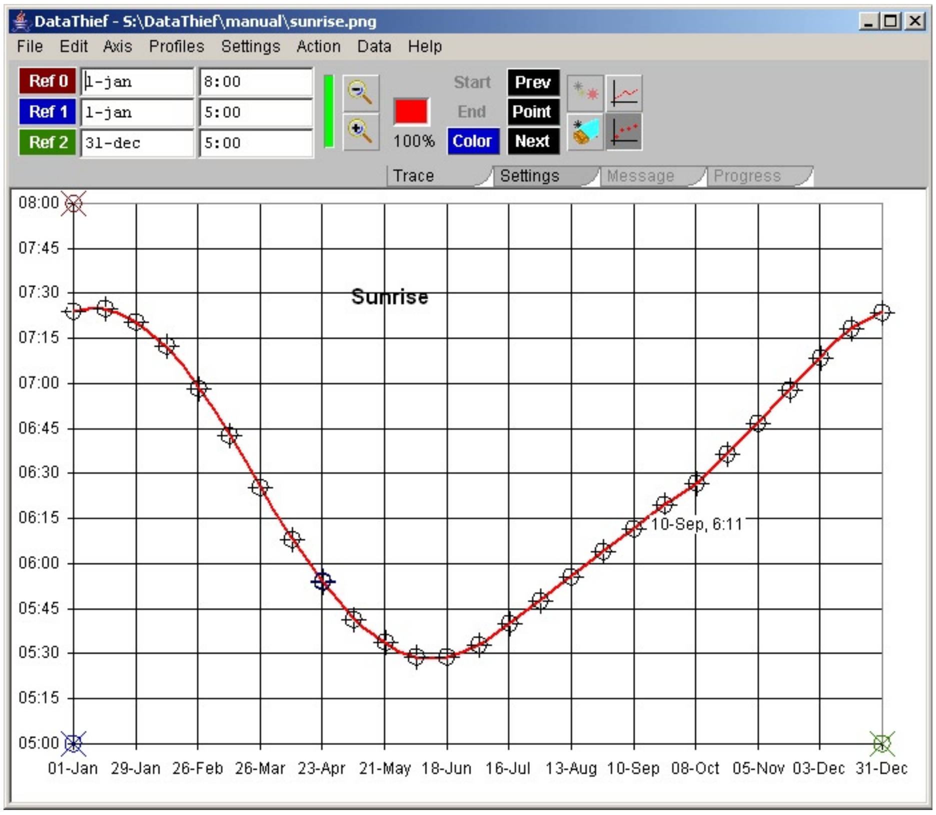
In this example the definition of "day" is already filled in (as the differences between 2 January and 1 January).

Click "Set" followed by "OK"

Now under the "Settings" tab you can specify

```
        Output distance     14 * day    on the primary axis
```

Trace the graph and enter Points mode.


DataThief - S:\DataThief\manual\sunrise.png
File Edit Axis Profiles Settings Action Data Help
Ref 0 1-jan 8:00
Ref 1 1-jan 5:00
Ref 2 31-dec 5:00
100%
Start End Color
Prev Point Next
Trace Settings Message Progress
Sunrise
10-Sep, 6:11
08:00
07:45
07:30
07:15
07:00
06:45
06:30
06:15
06:00
05:45
05:30
05:15
05:00
01-Jan 29-Jan 26-Feb 26-Mar 23-Apr 21-May 18-Jun 16-Jul 13-Aug 10-Sep 08-Oct 05-Nov 03-Dec 31-Dec

If you export the data, the translated values will be written out, so one line could read:

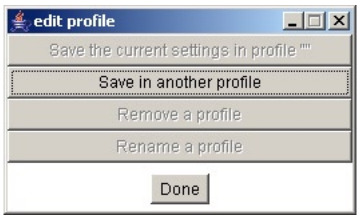
"10-sep", "6:11"

2.6. Profiles

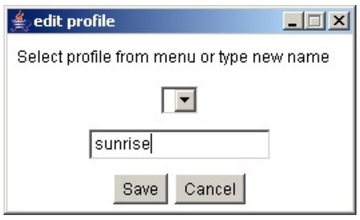
By default DataThief starts up with the settings as they were the last time DataThief was quit. That means that, once you have opened and image, all the settings were as you left them the last time.

This is not always desirable. It is quite possible that you have some graphs that have similar settings. To facilitate the processing hereof, it is possible to save all settings into a profile. When these specific settings - or setting rather similar to them - are needed, you van fetch them back by name.

To save a profile, you first set it up, meaning that you take care that all settings are as you would like them in the profile. Once that is done, you enter the profile manager: Profiles > edit profiles.



Because there are initially no profiles defined, the options to change, rename or remove a profile are not available. The only option you have is to save the current settings into a new profile. If you click "Save in another profile", the content of the profile editor changes to



Again, because no profiles are defined, the drop down menu is empty. All you can do is type in the name you wish to give this profile, then click "Save".

Now, the options to rename, renew or delete a profile will also be available.

DataThief has one profile that is not visible in this dialog: The settings when you last quit DataThief. By default these are the settings with which DataThief starts up, but under Edit > Preferences >General, you can select a defined profile to use when DataThief starts up.

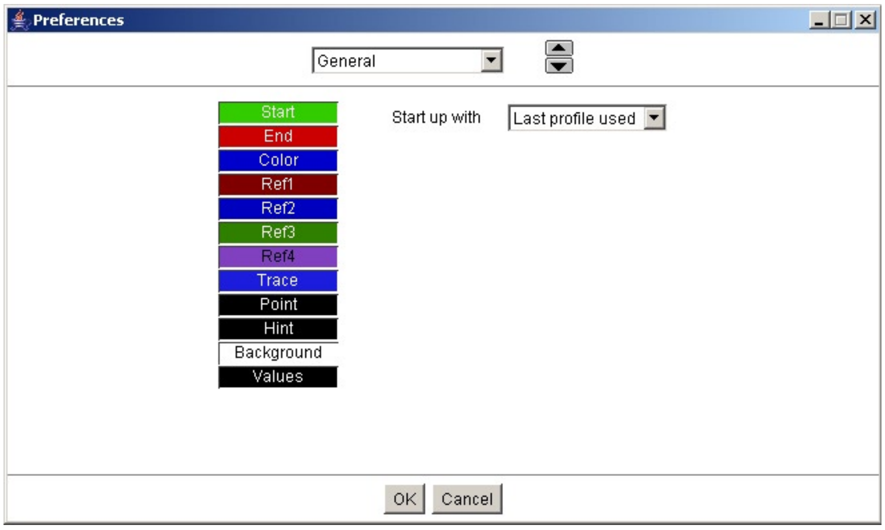
2.7. Preferences

Most of the possible settings of DataThief have already been covered. In this chapter we will nevertheless cover all of them again.

You can enter a specific preference panel by selecting the sub menu from the preferences item under the Edit menu.

Once in the Preferences panel, you can step through all preferences panels using the up and down arrows in the upper part of the window, or you can go directly to a specific panel using the drop down menu in the upper part of the window. With some exceptions, which will be explained with the appropriate panel, all settings will only be effectuated after you click the "OK" button at the bottom of the window. So, you may change colors, trace parameters, mouse clicks and data definition settings, than click the "Cancel" button and none of the changes will become effective.

2.7.1. General

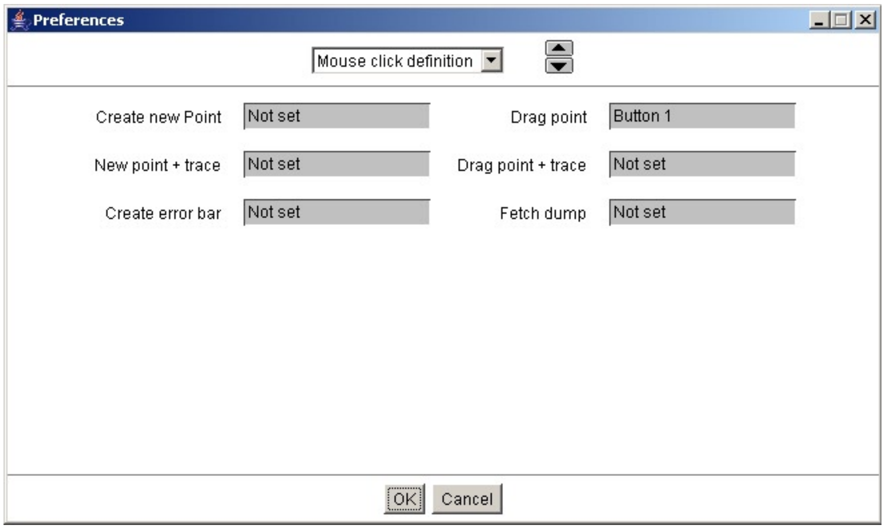


Start up with: Last profile used

In this panel you can set the profile to use when DataThief starts up, and you can set the colors for indicators.

With Start, End and Color you set the color of the Start- End- and Color indicators. With Ref# you set the color of a reference indicator. Trace defines the color of the traced line in Trace mode. Point sets the color of a point in Points mode. Hint sets the color of a Hint in Trace mode. Background defines the background color of the coordinate values shown next to a point in Point mode and Values defines the color of the text of those values.

2.7.2. Mouse click definition



Create new Point	Not set	Drag point	Button 1
New point + trace	Not set	Drag point + trace	Not set
Create error bar	Not set	Fetch dump	Not set

This panel allows you to configure your own preferences in mouse clicks. By default only "Drag point" is set to a (left) mouse click + drag.

You can define the clicks by clicking a mouse button, with your preference of modifiers keys (Shift, Control, Alt) pressed, in the gray area behind the text specifying the kind of action.

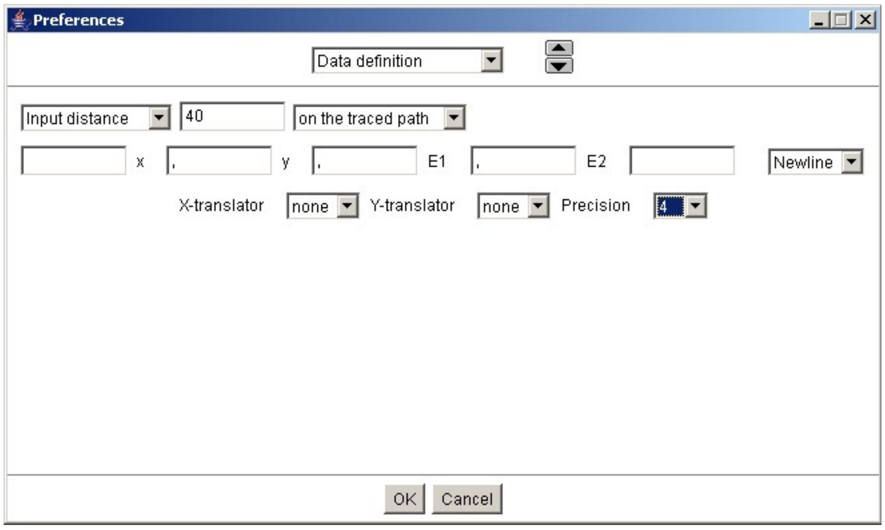
Create new point and Drag point may have the same click.

New point + trace and Drag point + trace may have the same click.

All other clicks must be different.

To clear a click action, click in the gray area with the same click as it has currently defined.

2.7.3. Data definition



Input distance 40 on the traced path

x , y , E1 , E2 Newline

X-translator none Y-translator none Precision 4

Here you specify which points to select after a trace, and the format to show and exports points.

Selection of points after a trace can be based upon distances measured in pixels (Input distance) or in terms of output coordinates(Output distance). The distance can be given as a number, or as an expression. Hence it is permitted to request an output distance of 2 * pi (provided pi is a defined value). The way distances are measured can be one of "On the primary axis" (usually the X-axis) or "On the traced path" meaning that distances are added until the requested value is exceeded.

The format of the text in files with exported data is defined in the next line. The order of data is always

value om primary axis, value on secondary axis, first error value, second error value

The error values are always positive. So a line reading

5.6, 3.5, 1.2, 0.7

can be interpreted as x-coordinate 5.6, Y-coordinate 3.5 with an error range between 3.5 + 1.2 and 3.5 - 0.7

If only one error value is given, you are dealing with symmetric error bars, so a line reading

5.6, 3.5, 1.2

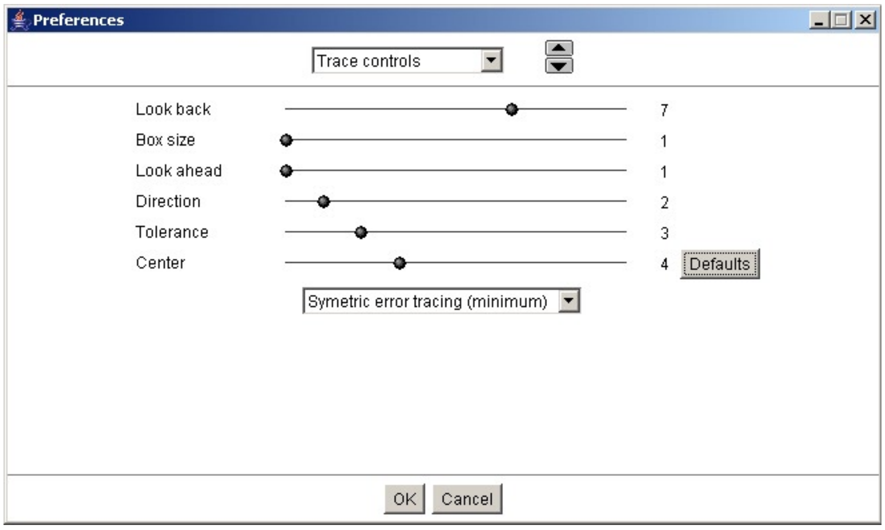
can be interpreted as X-coordinate 5.6 Y-coordinate 3.5 ± 1.2.

With only two values, you are dealing with data without error bars.

The formatting of the data is defined by the text fields surrounding the fixed parts of the exported data.

If a specific value is present in the exported data, then the content of the text field preceding it will be shown as well. The last two fields of the line are also always present.

2.7.4. Trace controls



Look back	7
Box size	1
Look ahead	1
Direction	2
Tolerance	3
Center	4

Symetric error tracing (minimum)

Here you define the parameter that affect the trace process.

Look back affects the number of points used to define the current direction. The bigger the number, the farther back the point, resulting in more reluctancy to take corners.

Box size affects the size of the square in which to look. A bigger box size allows for bigger gaps to jump, but can also result in greater computation time and a tendency to jump to other lines in the graph. The number you give here is just an indication to DataThief. Actually, the real size of the box is determined using this value, the value of "Look ahead - see following - and the thickness of the line to be scanned.

Look ahead affects the importance of nearness of the next point. The bigger the number, the less important it is that the next point is near to the current point, but DataThief will have a greater tendency to ignore intermediate points.

Direction affects the importance of maintaining the last direction (as defined by "Look back"). A bigger number will result in a reduced tendency to take turns.

Tolerance affects the deviation from the requested color that is allowed. The bigger the number, the more deviation is allowed.

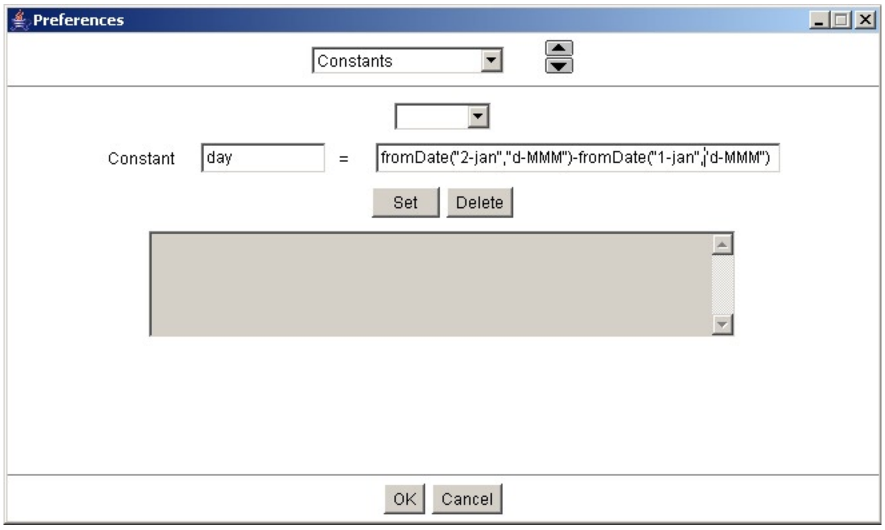
Center affects the stress that is laid on being near the center of the traced line. With thin lines this has no effect, but with thicker lines an increased number will steer the traced line more through the center, sometimes overriding all other aspects, resulting in a U-turn.

The last item is a pop up menu allowing you to define how error bar tracing in Points mode is performed. The possible settings are

• No error tracing	No automatic error tracing is performed,
• Symmetric error tracing (minimum)	Trace symmetric error bars, using the shortest of the positive and

	negative error bar as value for both.
• Symmetric error tracing (average)	Trace symmetric error bars using the average value.
• Symmetric error tracing (maximum)	Trace symmetric error bars, using the greatest of both values.
• Asymmetric error tracing	Trace asymmetric error bars.

2.7.5. Constants



Constants

Constant day = fromDate("2-jan","d-MMM")-fromDate("1-jan","d-MMM")

In this panel you can define Constant values for DataThief. Constants may seem a little misleading, as it is possible to define the constant is rather dynamic terms (such as "the current time"). But, once the value is defined, it will stay fixed until it is redefined. (With a constant defined as "the current time" that is until you start DataThief again).

To use more volatile values, you can define a function. See the next chapter.

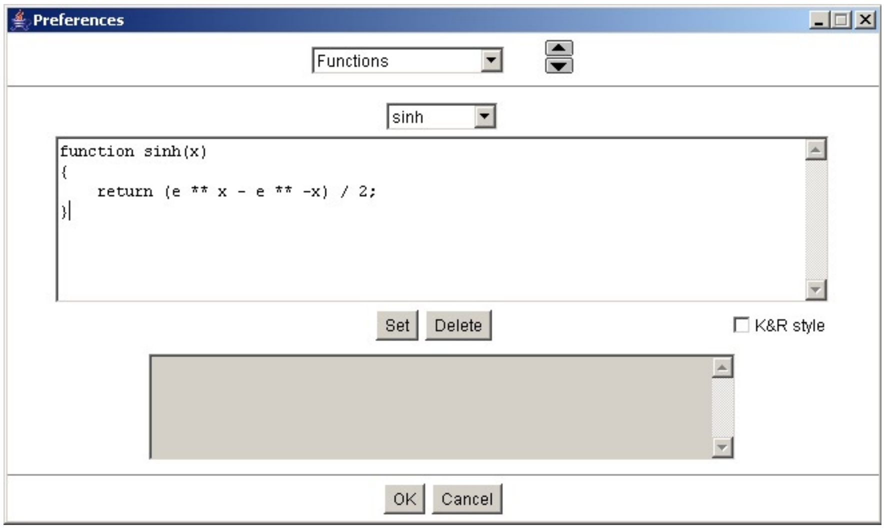
You define a constant by typing it's name in the text field after the word Constant, and it's value after the = sign. If you type the name of a constant that is already defined, you alter the value of that defined constant. The names of constants are case sensitive, hence PI and pi are different constants. To inspect (or change) the value of a constant, you select the name of the constant from the drop down menu, or you type the name of the constant in the text field after the word Constant.

The gray text area is used to display error messages in the definition of you constant.

Beware, that the changes become effective only after you press the "Set" button. Also be aware, that constants are define apart from clicking the "OK" button. Hence, you can make changes in the definitions of your constants and then click "Cancel", but the changes you made in the constants will not be cancelled.

To remove the definition of a constant, select the name of the constant from the drop down menu, or type the name of the constant in the text field after the word Constant, then press the "Delete" button. You will be asked confirmation. If you confirm that you want the constant removed, it is removed. Later applying Cancel will not bring the constant back.

2.7.6. Functions



```
function sinh(x)
{
    return (e ** x - e ** -x) / 2;
}
```

In this panel you can define any function you need. Most common use will be to define a function you need to specify a translator, or as part of the computations for a system of axes, but there are some functions with a special meaning.

The definition of a function is DataThief is much like defining a function in C or JavaScript. Basically a function definition is always

```
function name(parameterlist)
{
   // function body
}
```

The result of the function can be specified using "return". A function may contain more than one return statements. If the function reaches the end of it's body without encountering a return statement, the result of the function is undefined.

A definition for a function returning the absolute value of it's one parameter could be

```
function abs(n)
{
   if(n < 0)
      return -n;
   return n;
}
```

Though this can also be written as

```
function abs(n)
{
   return n < 0 ? -n : n;
}
```

For a full description of how to define functions, see chapter 2.9. DTC, Datathief Code.

Functions are automatically formatted. You have little choice in the result of formatting: indentation is always 4 spaces. comments always start on a new line, as does each statement. You can however choose the position of the curly braces that surround function bodies and compound statements. I prefer a style that uses lots of newlines, but puts the open and the close brace on the same horizontal position. The for anyone who knows C familiar - "Kernighan & Ritchie-style" (K&R-style) is more compact. You can choose between the two with the K&R-style check box.

To define a function, type it's definition into the central text area and click "Set". If you made a syntactical error in the function definition, the error message will be displayed in the gray text area and the function will not be defined. When no errors are found, the function is defined and added to the list of functions.

To change the definition of a function, select the name of the function from the drop down menu. The function definition will appear in the text area. You can change the definition then click "Set". Alternatively, you can just type the entire definition into the text area and type set. If you define a function with a name that already exists, the older function definition is overridden.

To delete a function definition, select the name of the function from the drop down menu, click "Delete", and click "OK" in the window that requests confirmation.

Be aware, that function definitions are set immediately, hence deleting a function, than regretting it and clicking "Cancel" will not bring the function back!

2.8. Menus

This chapter will give an overview of the menu structure of DataThief.

2.8.1. File

Open...	Allows you to open a file. Files are treated differently, depending on the file's extension: A file with the extension .dtf will be treated as a DataThief file, the result of a previous "Save". A File with the extension .dtc will be treated as a DataThief Code file. Global variable definitions will be added to the list of constants, and function definitions will be added to the list of functions. All other files are treated as files with graphical data. If DataThief is unable to convert the data to an image, an error message will be displayed.
Save	Save the current image with all it's settings in the current .dtf file. If you had not opened a .dtf file, but an image file. This results in the same action as "Save as...".
Save as...	Save the current image with all it's settings into a new .dtf file. You will be asked to enter the directory and the name of the file. If the file name does not have the correct extension. ".dtf" will be appended to the file name.
Export data	Save the collected data into a file. DataThief will ask for the directory and the name of the file. You must give the file an appropriate extension. DataThief will not give the file an extension, though it will suggest as file name the name of the current image with ".txt" as extension. The data will be saved in the format as defined by the settings in Edit > Preferences >Data definition.
Quit	This terminates DataThief. The current settings are saved in your preferences file, to be reused if you have declared In your Preferences that you wish DataThief to start up with the last settings.

2.8.2. Edit

Cut, Copy and Paste	These are not used in DataThief.
Preferences	Allows you to define your preferences. See Chapter 2.7.

2.8.3. Axis

This menu contains a list of all defined axes systems. As the list id ordered alphabetically, I suggest that you give you axes systems names that start with a number that allows you define the order. (See the predefined axes systems that come with DataThief). The last item of the menu is:

Edit Axes	Allows you to change, add or delete axes systems See chapter 2.2.

2.8.4. Profiles

This menu contains a list of all profiles you have saved. The menu is sorted alphabetically. The last item is:

Edit profiles	This shows a window allowing you to add or remove profiles. See chapter 2.6.

2.8.5. Settings

This menu contains items that modify the things you see in DataThief.

Show tool bar	The tool bar is the upper part of the DataThief window, containing the Reference point indicators, the zoom buttons et cetera. Once you have defined your system, you do not really need to see it, so you can use this menu item to hide or show

	the tool bar.
Show indicators	Once you have defined your system, the location indicators can work confusing. This menu item allows you to show or hide all location indicators. Note, that when the location indicators are hidden, the point indicators are disabled.
Show dump	This menu item allows you to show or hide the dump.
Values on MouseOver	This menu item affects the way values of points and location indicators are shown. If this item is checked, the coordinate values of a point are only visible when you move the mouse pointer over the point or indicator. If the item is not checked, all points and indicators show their coordinate values continuously.
Points mode	This menu item switches between Trace mode and Points mode.
4Point ref	This menu item allows you to select between the standard 3 reference points each with a coordinate both on the primary - and on the secondary axis, and the 4 reference point mode, where two reference points only define a Y-coordinate and the other two only define an X-coordinate.

2.8.6. Action

Zoom out	Zooms a factor 2 out of the image.
Zoom in	Zooms a factor 2 into the image.
Trace	Trace the current line (See chapter 2.3).
Reset	Move the location indicators to their default positions.
Enter key	This option is only present when you have not yet paid your shareware fee. When you pay your shareware fee, you receive a key to unlock all features. You can use this menu item to get a dialog where you can enter the key.

2.8.7. Data

Previous point	Show the previous point (in Point mode)
Previous hint	Show the previous hint (in Trace mode)
Current point	Show the current point (in Point mode)
Current hint	Show the current hint (in Trace mode)
Next point	Show the next point (in Point mode)
Next hint	Show the next hint (in Trace mode)

These items allow you the step quickly through all points or hints in your graph. The order that defines what is "Previous" or "Next" is defined by the order you entered points, or by the Trace process. To decrease the possibility of RSI, these items can also be selected by simple key clicks: p for Previous, n for Next and period for the current point or hint. It may be that you first have to click in the image area of DataThief for these key click to work.

Clear data	This item removes the traced line in Trace mode, or all points in Point mode.
Clear hints	This menu item removes all hints in Trace mode.

2.8.8. Help

About DataThief	Shows DataThief's About dialog

2.9. DTC, DataThief Code

2.9.1. Introduction

DataThief Code is a simple language that allows you to define almost any conversion you need that is not predefined in DataThief.

The language takes many elements from C and from JavaScript, but is certainly not intended as a general purpose tool. It is intended as a tool to allow you to create functions to facilitate the definition of a system of axes, and to define data translators.

2.9.2. Lexical structure

DTC is case sensitive. Hence "while" is a keyword, en "While" is not. The predefined function sin() can not be invoked with "SIN(1.2)".

DTC is parsed using a "longest match first" method. This means that in ambiguous situations, the possibility with the most matching characters is used. As an example:

DTC has an increment operator `++` that can be applied to a variable. `a++` means "set the value of `a` to `a+1`.

DTC has a + operator that take two values or variables as operand. a + 1 means exactly what you would think.

But combining these two, it is correct DTC to write `a+++b`

DTC will interpret this as `a++ + b` and not as `a + ++b` which is also correct DTC.

Of course it is advised not to rely on this behavior; it makes your code difficult to read and error prone.

Any text between // and the end of a line is DTC comment, and is ignored.

An identifier is the name of a variable of function in DTC. As in JavaScript or C, DTC identifiers must begin with an upper- or lower case letter or and underscore (_). Subsequent characters may be letters, digits or underscores. There are a number of reserved words that fulfill this definition, but nevertheless cannot be used as identifier. These words are

```
array break case catch char continue default else float for function if int local
return sizeof string switch try typeof while
```

2.9.3. Data types

DTC supports integer numbers, floating point numbers, strings, characters and arrays.

Each of these will be discussed in the rest of this chapter.

2.9.3.1. Integer

An integer, int for short, is a signed whole number with a value in the range of -9223372036854775808 and 9223372036854775807. Arithmetic operations that exceed these limits will generate an error condition.

Integer literals are sequences of digits. For example

```
0
7
13684
```

You can specify hexadecimal integer literals by preceding the sequence by 0x or 0X. The subsequent sequence may also contain the letters from a to f or from A to F to denote the hexadecimal values from 10 to 15. Hence to denote the decimal value 110 in hexadecimal you may write

```
0x6e
```

An octal denotation of an integer is specified when you precede the integer literal by 0. Hence

```
035
```

is 29 in decimal.

To specify a negative integer you may precede the integer literal by a minus sign (-). But, as the minus is not part of the literal, but is a unary operator, white space between the minus sign and the integer literal is allowed.

As a minus sign is interpreted as unary operator, the number -9223372036854775808 can not be expressed as a literal integer because the positive value 9223372036854775808 can not exist. In the rare event that you need this literal value you can use the hexadecimal equivalent 0x8000000000000000.

2.9.3.2. Float

A floating point number, float for short, is a number that may have a decimal point, and an exponent part. Floats are written the same way as in C and JavaScript: an optional integer part, an optional decimal point, an optional fraction, and an optional exponent part consisting of a lower or upper case E followed by a signed integer. Valid floats are:

```
1.
.1
53.954
4e2
1345.5e12
1.602189E-19
```

As with integers, floats can be made negative using the unary minus operator.

2.9.3.3. String

A string is a sequence of characters. Unlike in C, a DTC string is not zero terminated. A string can contain a character with the value 0.

A literal string is any sequence of characters except newline enclosed by double quotes ("). Like C and JavaScript, certain characters can be escaped within a literal string. This is done by preceding the escaped character by a backslash (\). The meaning of escaped characters is:

\b	backspace
\n	newline
\r	return
\\\\	backslash
\\"	"
\\###	where ### one to three octal digits: the ASCII character corresponding to the digits
\c	where c is any other character: the character as if no \ was present

Valid string literals are

```
"Hello world"
"quote \" "
"\102 is B"
```
(this is equivalent to `"B is B"`)

2.9.3.4. Characters

A character is an integer with very limited range: the value of a character is between 0 and 255. At the same time, a character can be seen as a string containing only one character. When a character takes part in an expression it is silently converted to an integer or a string, depending upon context, but basically, for expressions a charatcer is an integer. A character literal is the required character surrounded by single quotes '. You can use the same escape sequences as in strings.

Examples of character literals:

```
'a'
'\r'
'\040'
'\''
```

2.9.3.5. Array

An array is a list of values that can be accessed through an index. Arrays in DTC always begin at index 0.

Array cells can contain any type of value, so it is well possible that one array contain integers, strings and even arrays.

Arrays are sparsely filled. This means that cells that have not been given a value, nor have been inspected, have no value attached. If you inspect the cell without giving it a value, the cell is filled with the integer value 0.

A literal array is a comma separate list of values surrounded by square brackets. For example

```
[1, 1, 2, 3, 5, 8, 13
[1, 3.14, "one", [1, 2, 3]]
```

Array cells are accessed by giving the array followed by the index (counting from 0) of the cell surrounded by square brackets.

Hence the result of

```
        [1, 1, 2, 3, 5, 8, 13][5]
```

is 8.

The values of a literal array may actually be expressions containing variables, but the value must be defined at the moment the literal array is parsed. As a result the construction

```
        for(i = 0; i < 5; i++) { a = [i] ...
```

Will raise an error "i not defined" because the literal array `[i]` is parsed before the for loop that defines it is executed. A simple work around uses `[]` to define an empty array

```
        for(i = 0; i < 5; i++) { a = []; a[0] = i ...
```

The same construction can be used to create an array that can be sparsely filled.

An example of the use of sparsely filled arrays is

```
        a = []; // create an empty array
        a[10] = 1;
        a[100] = 1;
```

Now the array attached to the variable a has a length of 101, but contains only two values.

```
          b = a[50];
```

After this statement, the array still has a length of 101, but now it contains three values, because examining the 51st element has created a value of 0 into the cell.

2.9.4. Variables

A variable is an identifier, that is associated with a value. In DTC variables are created by giving them a value. For example

```
          pi = 3.141592653589793;
```

Creates a variable called pi and gives it a value that is a reasonable approximation of π.

If later on in the code defies all known mathematics by stating

```
          pi = 3;
```

The value of the variable pi is changed. In paragraph 2.9.7.3 we shall see that it is possible for several variables with the same name to co-exist.

2.9.5. Expressions and operators

2.9.5.1. Expressions

An expression is a piece of code that eventually produces a value. The simplest expression is just a value:

```
        5
```

More complex expressions combine simple expressions. For example

```
        5 + 7.5
```

which of course results in a floating point value of 12.5.

The type of the result of an expression depends on the operator and it's operands. In the example above the operator plus with an integer left operand and a float right operand results in a float. Whereas applying the plus operand to two integer operands would yield an integer result.

2.9.5.2. Operator overview

The way an operator works is defined by it's precedence, it's associativity, the number of operands and it's operation.

Precedence defines how grouping of operators takes place. For instance the expression

```
        2 + 3 * 5
```

groups as

```
2 + (3 * 5)
```

because * has a greater precedence than +.

Associativity defines the order of evaluation of operators of the same precedence. Associativity can de Left-to-right of Right-to-left. Normal associativity is Left-to-right. For instance

```
2  - 3 + 5
```

groups as

```
(2 - 3) + 5
```

because + and - have the same precedence and are both Left-to-right associative.

Whereas

```
a = b = 3
```

results in both a and b getting the value of 3 because = is Right-to-left associative.

You can force grouping using brackets. Hence (`2 + 3) * 5` yields 25.

DTC has three kinds of operators:

unary operators, that have only one operand. For instance the negation operator -, when used in - 3.

binary operators that have two operands, such as the addition operator in 2 + 3.

There is one ternary operator ?: which combines three operands into one expression.

2.9.55.3. Unary operators

All unary operators have the same precedence and group from left-to-right.

- unary minus	Can be applied to an integer (remember that chars can be used wherever an interger is expected) or a float. The result is the same type with the value negated.
! logical not	Can be applied to an integer or float. The result is an integer. 1 if the operand was 0, 0 for all other values of the operand.
~ bitwise not	Can be applied to an integer. Returns an integer with each bit negated.
++ pre increment	Can be applied to a variable containing an integer or a float. The contents of the variable is replaced with it's value + 1. This new value is also the result of the operator. pre increment means that the operator is put before the identifier. For example `++val`

++ post increment	Can be applied to a variable containing an integer or a float. The contents of the variable is replaced with it's value + 1. The result of the operator is the old value of the operand. For example
	`val = 5;`
	`prod = 2 * val++;`
	has as result that the variable prod has the value 10 and val now has value 6;

-- pre increment	Can be applied to a variable containing an integer or a float. The value of the variable is decreased by one. The new value is returned.
-- post decrement	Can be applied to a variable containing an integer of a float. The value of the variable is decreased by one. The old value of the variable is returned.

sizeof	Can be applied to any type. The result it the number of elements of the operand. Hence for integer and float operands the result is 1. For a string the number of characters in the string is returned and for an array the size of the array; For instance
	`row = [];`
	`row[12] = 1;`
	`a = sizeof row;`
	results in a having the value 13. Whereas
	`row = [1, 1, 2, 3, 5, 8, 13];`
	`a = sizeof row;`
	results in `a` containing 7.

typeof	Can be applied to any operand. The result is a string containing "int", "float", "char", "string" or "array".
(type)	The cast changes the type of it's operand. The cast consists of the name of the target type surrounded by brackets. When you cast an integer to char, the number is

truncated silently to the range of 0 to 255. When you cast a float to integer of char, the integer part of the number is taken and silently truncated to fit the target type. You can cast integers and floats to a string, resulting in a string containing the literal representation of the number. If you cast a character to a string, you get a string containing one character. An array can only be cast to array (this does nothing but consumes processing power). Examples of casts:

```
(float)1        // => 1.0
(int)1.4        // => 1
(int)1.9        // => 1
(char)102       // => 'f'
(string)'f'     // => "f"
(string)102     // => "102"
```

2.9.5.4. Binary operators

The precedence of binary operators is

[]	subscript
**	to the power
*, /, %	multiply, divide, modulo
+, -	add, subtract
<<. >>, >>>	shift left, shift right, shift right zero fill
<. <=, >, >=	less than, less than or equal, greater than, greater than or equal
==, !=	equals, not equals
&	bitwise and
^	bitwise exclusive or
|	bitwise or
&&	logical and
||	logical or
..	slice

These all group left-to-right.

Assignment operators all group right-to-left and have the same precedence.

.. slice

A slice can only be used within a subscript (see the next paragraph). Left and right hand operand of the slice must be integer values. The final result of the slice must be such, that the right hand operand is not smaller than the left hand operand, but each of the operands may be negative, indicating that the operand is added to the size of the array or string that is subscripted. See the next paragraph for examples.

[] subscript

The subscript is a slightly weird operator. It is the only operator that has a part of the operator right of the right operand. But it reads quite naturally.

The left hand side of the subscript is an array or a string. The right hand side is a character or a slice (see the previous paragraph)

If the right hand side operand is a slice, the result of the subscript is the same type as the left hand side operand, i.e. a slice from an array is an array and a slice from a string is a string.

If the right hand side operand is an integer, the result is the type of the cell that is taken for in array, and the result is char for strings. This char's value is the ASCII value of the character taken from the string. For arrays the result of an integer subscript is the value and the type of the designated cell.

For example:

```
a = [ 2, 3, 5, 7, 11, 13, 17, 19, 23 ];
a[3]  => 7
a[3..3]  => [ 7 ]
a[3..5]  => [ 7, 11, 13 ]
a[3..-3]=> [ 7, 11, 13, 17]
w = "abcdefghi";
w[3]  => 'd'
```

```
w[3..3]  => "d"
w[3..5]  => "def"
w[3..-3] => "defg"
w[-3..-1] => "ghi"
```

Note, that because the slice operator has a very low precedence, constructions like

```
w[i .. i + 2]
```

will indeed take a slice of three elements from the ith element.

**** to the power**

Operands can be float or integer. The result is always a float. Be aware that this operator may raise an error condition. For example `-2 ** .5` would attempt to take the square root of -2.

*** times**

Operands can be float or integer. The result is integer if both operands are integer, otherwise the result is float. If both operands are integer an error condition "integer overflow" can occur when the result exceeds the possible values for integers.

/ divide

Operands can be float or integer. When both operands are integer, the result is an integer with as value the whole part of the division. E.g. `18 / 5` results in 3 whereas `18.0 / 5` has 3.4 as result.

% modulo

Both operands must be integer. The result is the remainder of the division of the right hand operand by the left hand operand.

For example

```
18 % 5  => 3
-18 % 5 => -3
```

+ addition

When both operands are integer, the result is an integer. When both operands are integer of float, the result is a float.

String + string results in a string where both strings are concatenated.

String and numbers can be added. The number is first converted to a string.

Finally, two arrays can be added, resulting in an array with all elements of both arrays

Examples:

```
2 + 3.4 => 5.4
2 + "ok"=> "2ok"
"ok" + 2=> "ok2"
"a" + "b"  => "ab"
[1, 2] + [3, 4, 5] => [1, 2, 3, 4, 5]
```

Sparsely filled array that are added will result in a new sparsely filled array.

- subtraction

Operands may be integer or float. The result is integer when both operands are integer. Otherwise the result is float.

<< shift left

Both operands must be integer. The result is the bit pattern of the left operand binary left shifted by the number given by the right operand. For example

```
5 << 2  => 20 (5 is 101, 20 is 10100)
```

>> shift right

Both operands must be integer. The result is the bit pattern of the left operand binary right shifted by the number given by the right operand. Following the previous example

```
20 >> 2 => 5
```

but

```
        -1 >> 2 -> -1
```

This is because the left most bit of -1 is a 1 (actually all bits of minus 1 are 1). As the >> operator replicates the left most bit (and discards the right most bit) shifting -1 right results in -1.

>>> shift right zero fill

As shift right, but in stead of replicating the right most bit, the right most bit is set to 0; This is equivalent to an unsigned right shift.

< less than, <= less than or equal, > greater than, >= greater than or equal, == equals, != not equals

Operands can be any combination of float, integer and string. If one of the operands is string, the other is converted to a string and the comparison is alphabetically lexicographic. For integers and floats the comparison is numeric. The result is integer 1 if the comparison yields true, integer 0 otherwise. Examples:

```
        1 < 2.0 => 1
        "1" < "2"  => 1
        11 < 101=> 1
        11 < "101" => 0  ("11" comes lexically after "101")
```

& bitwise and

Both operands must be integer. The result is an integer that has a 1 bit where both operands had a 1 bit, and a 0 bit where at least one of the operands had a 0 bit. Take for example

```
        0xa9 & 0x27

        0xa9 =  1010 1001
        0x27 =  0010 0111
        result  0010 0001
```

^ bitwise exclusive or

Both operands must be integer. The result is an integer that has a 1 bit where both operands had non matching bits, and a 0 bit where at both operands had equal bits. Take for example

```
        0xa9 ^ 0x27

        0xa9 =  1010 1001
        0x27 =  0010 0111
        result  1000 1110
```

| bitwise or

Both operands must be integer. The result is an integer that has a 0 bit where both operands had a 0 bit, and a 1 bit where at least one of the operands had a 1 bit. Take for example

```
        0xa9 | 0x27

        0xa9 =  1010 1001
        0x27 =  0010 0111
        result  1010 1111
```

&& logical and

Operands can be integer or float. The result is an integer 1 if both operands are unequal to 0. If either operand is 0, the result is 0. It is guaranteed that the second operand is not evaluated if the first operand equals 0. So the effect of

```
        a = 1;
        c = 0;
        c && a++;
```

is that `a` holds it's value of 1; the `a++` is not performed.

|| logical or

Operands can be integer or float. The result is an integer 1 if either operands is unequal to 0. If both operand are 0, the result is 0. It is guaranteed that the second operand is not evaluated if the first operand does not equal 0.

Assignment operators

The primary assignment operator is =.

= is both used to declare a variable, and to modify the value of an existing variable.

Because assignment operators group right to left, it is possible (though not very readable) to state

```
        a = 4 + b = 7;
```

which is equivalent to

```
        b = 7;
        a = 4 + b;
```

All other assignment operators are of the type "ue and assign". They are all written as the operator to be used immediately followed by an = followed by the second operand. The result is that the operator is applied to both operands and the result is stored in the left hand operand, which must be a variable. Take for example the "added by" +=.

```
        a = 3;
        a += 4;
```

results in a having a value of 7.

As these operators evaluate right to left, it is possible to write

```
        a = 3;
        b = 4;
        a += b += 5;
```

Which is equivalent to

```
        a = 3;
        b = 4;
        b += 5;
        a += b;
```

The following use and assign operators exist

`*= /= %= += -= <<= >>= >>>= &= ^= |=`

The left hand side operand of an assignment operator can be a subscription. So it is valid to write

```
        s = "abc";
        s[1] = 'x';   // s now contains "axc"
        s[0]++;       // now s is "bxc";
        r = [2, 3, 5, 7, 9];
        r[1] += 5;
```

2.9.5.5. Ternary operators

There is one ternary operator. This functions as an in line if statement, The operator looks like

```
        condition ? truevalue : falsevalue
```

Condition can be integer of float. truevalue and falsevalue can be any type, and need not even be of the same type. As the precedence of the ? : operator is very low (just above the level of assignment operators) you will often need brackets to obtain the desired result. For example to match the singular or plural to count you could write

```
        "I have " + count + (count == 1 ? " egg" : " eggs")
```

or

```
        "I have " + count + " egg" + (count == 1 ? "" : "s");
```

But if you forget the brackets, the result is the string `"s"`, because this groups as

```
        (("I have " + count + "egg" + count) == 1) ? "" : "s";
```

2.9.6. Statements

Statements are the sentences of DTC. Most statements in DTC end with a semicolon.

2.9.6.1 Simple statements

The simplest statement that DTC knows is the "empty statement"

```
;
```

It does absolutely nothing (which is sometimes what you want).

Then, every expression is a simple statement (even when the expression is not simple at all).

2.9.6.2. Compound statements

A compound statement is a group of statements surrounded by curly braces {}. A compound statement can be used where DTC requires simple statement and you need more than a simple statement.

Compound statements do not end with a semicolon.

2.9.6.3. If, else

The if statement allows you to choose between alternatives. The form of an if statement is

```
if(condition) statement else statement
```

As it frequently happens that you do not require the `else statement` part, you may omit the `else ;`

So a short form of the if statement is

```
if(condition) statement
```

The brackets around the condition are required.

Examples of if statements are

```
if(n < 0) n = -n;
```

which implements a simple absolute value.

```
sign = 1;
if(n < 0)
{
   n = -n;
   sign = -1;
}
```

which does the same, but remembers the original sign. Note the use of a compound statement.

```
if(n < 2)
   plural = 0;
else
   plural = 1;
```

This looks maybe a bit like the example used with the ?: operator, but remember that the if statement does not have a result, so you can not write

```
a = if(n < 1} "x"; else "y";
```

This will result in an error message "Illegal statement".

2.9.6.4. Switch

If you need to choose out of more than two options, you can use the switch statement.

The syntax of the switch statement is:

```
switch(expression) statement
```

But you will actually always need a compound statement.

One of the statements you can use within a switch statement is a case label.

The case label looks like

```
case expression:
```

When a switch statement is encountered, the value of the expression of the statement is computed. Then the (compound) statement is searched for a case label with an expression that evaluates to the same value. All statements following that case label are executed until the end of the compound statement or until a break statement.

A break statement is

```
        break;
```

If no matching case label is found, but there is a default label in the compound statement

```
           default:
```

execution resumes at the default label again until the end of the compound statement, or until a break statement.

Note that encountering another case label or default label does not interrupt the execution of statements.

A simple example:

```
        switch(v)
        {
           case 0:
              s = "zero";
              break;
           case 1:
              s = "one";
              break;
           case 2:
              s = "two";
              break;
           default:
              s = "???";
        }
```

It is possible to combine several conditions within one action, or have a condition use only a part of an action as in

```
        switch(value)
        {
           case 1:
              s = "ok";
           case "2":
           case 3:
              b = 4;
              break;
           default:
              b = 10;
        }
```

In this example, the statement `b = 4;` is performed when value equals 1, "2" or 3. `s = "ok"` is only performed when value equals 1. Note the mixed use of types in the case labels. The test performed to match case labels is the same as for the == operator, meaning that if either operand is string, the other operand is converted to string, so actually we could also have written `case 2:`. It is allowed, though not useful to have more labels with the same value, or more default labels. program execution is started at teh first label that matches expression, or at the fist default label, and continues unti the end of the compund statement or until a break statement.

2.9.6.5. While

The while statement allows you to repeat actions. The form of a while statement is

```
        while(condition) statement
```

The statement, that is usually a compound statement and called the body of the loop, will be repeated as long as condition yields a result not equal to 0. For example

```
        count = 0;
        sum = 0
        while(count < 10)
        {
```

```
      sum += count++;
   }
```

Will leave the value of sum at 55 and the value of count at 10. Note the use of the post-increment operator.

It is possible to leave the while statement somewhere in the middle of the body. The statement `break` allows you to jump out of the while statement. For example

```
   count = 0;
   result = 0;
   while(count < 100)
   {
      result += ++count;
      if(result > 100)
      {
         break;
      }
      result++
   }
```

will end when count equals 100 or result is greater than 100 whatever comes first.

The break statement jumps only out of one while loop, so when you have nested while statements as in

```
   i = 0;
   while(i < 10)
   {
      j = 0;
      while(j < 10)
      {
         if(i > j)
         {
            break;
         }
      }
      // execution resumes here after the break
   }
```

It is also possible to jump immediately back to the test of the condition without executing the rest of the body of the loop. This is done with the `continue` statement. As with the break statement, continue only affects the innermost while statement.

```
   count = 0;
   result = 0;
   while(count < 100)
   {
      result += ++count;
      if(result > 100)
      {
         continue;
      }
      result++
   }
```

will not perform the `result++` once `result` exceeds 100.

2.9.6.6. For

The basic structure of a while loop is often:

initialize condition; while(test condition) { do things; change condition; }

This is so common, that there is a statement that does exactly that: the for loop.

```
   for(initialize; condition; change) statement
```

`initialize`, `condition` and `change` are (possibly empty) expressions. A simple example:

```
        sum = 0
        for(i = 1; i <= 10; i++)
        {
           sum += count;
        }
```

You can use `break` and `continue` statement in the for loop. The break jumps out of the loop. The continue executes the `change` expression then dependent on the `condition` enters the body again, or terminates.

All three expressions may be empty, so forever can be written as

```
        for(;;);
```

Not very useful maybe, but once you add a body with a break, it could make sense.

2.9.6.7. Try, catch

Usually, when DTC encounters an error condition, such as `integer overflow`, or `array index out of bounds` execution of the program is stopped, and DataThief shows an error message. Sometimes that is not what you want, so it is possible to catch the error message. The syntax is

```
        try statement1 catch(variable) statement2
```

When an error occurs in the execution of `statement1`, `variable` will receive the error message and statement2 is executed. For example

```
        s = "abc";
        try
        {
           v = s[4];
        }
        catch(error)
        {
           // eror will contain String "String index out of bounds"
           v = 0;
        }
```

The effect is that whenever s contains less then 5 characters, or when s is not a string or array, v will contain 0;

2.9.7. Functions

Almost all information concerning DTC has been given to enable you to write your own functions. A function is little more than a compound statement that returns a value. But the nice thing is, that even though you can not use a compound statement in an expression, you can use a function in an expression.

This allows you to add more mathematics to your DataThief then I could ever think of.

2.9.7.1. Defining functions

A function definition has as syntax:

```
        function name(parameters) { statements }
```

Name may be any identifier. DataThief allows you to have a function with the same name as a variable or a constant, but this may make your code less readable.

Parameters is just a list of identifiers separated by comma's. When the function is invoked (called), you have to give as many values as you had identifiers in de function definitions parameter list.

In the function body you can use any of the parameters, or you can create your own variables, or use those that were already defined before the function was called. When the function has to return a value, you can use the return statement:

```
        return expression;
```

The expression after return may be empty, causing the function not to return a value, but it stops execution of the statements of the function and returns control to the calling environment of the function.

A very simple function definition is

```
function one()
{
   return 1;
}
```

Note that this function takes no parameters.

A slightly more complicated function computes the square of its single parameter

```
function square(value)
{
   return value * value;
}
```

Obviously, when this function is called with a parameter that is not integer or float, you raise an error. Suppose that no matter what, you do not want to see that error, but in stead you want the function to return 0. I shall show two alternatives.

```
function square(value)
{
   switch(typeof value)
   {
      case "char":
      case "integer":
      case "float":
         return value * value;
   }
   return 0;
}
```

But actually the following solution is faster

```
function square(value)
{
   try
   {
      return value * value;
   }
   catch(error)
   {
      return 0;
   }
}
```

This solution has as added feature, that in case of overflow 0 is returned.

As an example of a function with more parameters a function that returns the solution of a quadratic equation in an array containing 0, 1 or 2 values.

```
function solveQuad(a, b, c)
{
   // remember: (-b +/- root(b ** 2 - 4ac)) divided by 2a
   result = [];
   // if a == 0, it is linear
   if(a == 0)
   {
      if(b != 0)
      {
         result[0] = (float)c / b;
      }
      return result;
   }
   v = b * b - 4 * a * c;
   if(v < 0)
   {
```

```
            //no solutions
            return result;
        }
        if(v < 1e-20)
        {
            // v very small, only one solution
            result[0] = -b / (2.0 * a);
            return result;
        }
        v = sqrt(v);
        result[0] = (-b + v) / (2 * a);
        result[1] = (-b - v) / (2 * a);
        return result;
    }
```

Note that we use floating point numbers and once a cast to force the result of division to be floating point.

2.9.7.2. Invoking functions

A function is called by using it's name followed by a bracket, the parameters separated by comma's, and a close bracket. Hence the function square we defined in the previous paragraph could be called by

```
square(4)
```

But usually the function call will be part of an expression as in

```
v = 3 * square(4);
```

Note that even when the function has no parameters, the brackets must be present in the function call, so the function "one" that was introduced in 2.9.7.1 must be called with

```
a = one();
```

The parameters of a function are passed by value. This means, that if you use a variable as parameters and in the function you change the contents of the parameter, the value of the variable outside the function is not changed.

```
function mod(var)
{
    var = 2;
    return var;
}
a = 1;
mod(a);
// a is still 1
```

But the contents of parameter is not copied. This means that if you have a string or array as parameter, any changes to the contents of the parameter reflect outside the function.

```
function toUpper(s)
{
    if(s[0] >= 'a' && s[0] <= 'z')
    {
        s[0] += 'A' -'a';
    }
}
hi = "hello world";
toUpper(hi);
// hi contains "Hello world"
mod(hi);
// hi still contains "Hello world"
```

2.9.7.3. Scope of variables

Consider the following code

```
function possible_side_effect()
{
   a = 1;
   return a;
}
```

If this function is called when a constant `a` is in existence, this existing `a`'s value will be altered. This is because when a variable name in a function is encountered, DataThief searches in the body of the function for a variable with that name. It the variable is not found, DataThief searches the global level - within DataThief that are all constants - for a variable with the required name. I that is not found as well, a new variable is created within the body of the function. But, obviously, if the variable existed on the global level, that variable is used. This is usually not what you want. To prevent DataThief from searching the global level, or more precisely, to force DataThief to create a new variable within the body of the function, you can precede the variable name with the keyword "local".

So a new version of above function is

```
function no_side_effect()
{
   local a = 1;
   return a;
}
```

If you use the "local" keyword, you must initialize the variable, so a C-style or JavaScript style declaration

```
local a;
...
a = 1;
```

Is not valid.

How local is local?

The answer is: "Very".

consider

```
function a()
{
   local x = 1;
   for(local x = 3; x < 5; x++)
   {
      ...
      z = x;
   }
   local y = x;
}
```

At the end of the function y will have a value of 1. The local x in the for loop is indeed local to the loop. But, unless the variable z is defined globally, it is also local to the same for loop. If you needed z not to interfere with a possible global variable z, but needed the value outside the loop you must use a construction as in

```
function a()
{
   local x = 1;
   local z = 0;
   for(local x = 3; x < 5; x++)
   {
      ...
      z = x;
   }
   local y = x;
   // now z is known
}
```

2.9.8. Predefined functions

DataThief knows the following predefined functions

2.9.8.1. Conversion

deg

```
function deg(angle)
```

angle is in radians. The result is the angle in degrees.

rad

```
function rad(angle)
```

angle is in degrees. The result is the angle in radians

ceil

```
function ceil(value)
```

value is int or float. The result is the smallest int not smaller than value. E.g. `ceil(3.2) => 4`

floor

```
function floor(value)
```

value is int or float. The result is the biggest int not greater that value. E.g. `floor(3.2) -> 3`

round

```
function round(value)
```

value is int or float. The result is the int nearest to value. `round(3.2) => 3; round3.5) => 4`

join

```
function join(row, between)
```

This function joins all elements of the array `row` to a single string. The string `between` is interjected between each of the elements.

```
join(["Hello", "world", 1, 2], " --- ") => "Hello --- world --- 1 --- 2"
```

split

```
function split(str, between)
```

the string `str` is converted to an array. Each occurrence of the string `between` is used to split the string. If `between` is not present in `str`, the result is an array with one element which is the original `str`. If `between` is an empty string "", The result will be an array of strings, each containing one character of the original `str`.

```
split("This is a string", " ") => ["This", "is", "a", "string"]
split("This is a string", "q") => ["This is a string"]
split("This is", "") => ["T", "h", "i", "s", " ", "i", "s"]
```

A funny effect of split and join is that you can use it as a string substitute function:

```
function substitute(str, from ,to)
{
   return join(split(str, from), to);
}
```

2.9.8.2. Mathematical

sin, cos, tan

```
function sin(angle)
function cos(angle)
function tan(angle)
```

Return the sine, cosine and tangent of the int or float angle. angle is in radians.

asin, acos, atan

```
        function asin(value)
        function acos(value)
        function atan(value      )
```

Return the inverse trigonometric function. The result is in radians.

atan2

```
        function atan2(x, y)
```

Returns the arc tangent of x / y, using the sign of both to determine the quadrant. The result is in radians.

sqrt

```
        function sqrt(value)
```

Returns the square root of the int of float value.

log, ln

```
        function log(value)
        function ln(value)
```

Return the base 10 respectively natural logarithm of the int or float `value`.

2.9.8.3. Date and time

currentTime

```
        function currentTime()
```

Returns an integer containing the current time in milliseconds since January 1, 1970.

toDate

```
        function toDate(time, format)
```

This function converts the `time` to a string according to `format`.

Format may contain the patterns from the table below to insert specific parts of the date in the resulting string. Any character in format that do not appear in the table appear literally in the resulting string.

Field	Full form	short form
Year	yyyy (4 digits)	yy (2 digits)
Month	MMMM(name)	MMM (3 characters of name)
Month	MM (2 digits)	M (1 or 2 digits)
Day of week	EEEE	EE
Day of month	dd (2 digits)	d (1 or 2 digits)
Hour (1-12)	hh (2 digits)	h (1 or 2 digits)
Hour (0-11)	KK (2 digits)	K (1 or 2 digits)
Hour (1-24)	kk (2 digits)	k (1 or 2 digits)
Hour (0-23)	HH (2 digits)	H (1 or 2 digits)
Minute	mm (2 digits)	m (1 or 2 digits)
Second	ss (2 digits)	s (1 or 2 digits)
Millisecond	SSS (3 digits)	S (1, 3 or 3 digits)
AM/PM	a	
Time zone	zzzz	zz
Day of week in month	F (e.g. 3rd Thursday)	
Day in year	DDD (3 digits)	D (1, 2, or 3 digits)
Week in year	ww(2 digits)	w (1 or 2 digits)
Era (BC/AD)	G	

```
toDate(1000000000000, "d-MMM-yyyy HH:mm:ss") => "9-Sep-2001 03:46:40"
```

fromDate

```
        function fromDate(date, format)
```

This is the reverse of toDate. FromDate takes the string `date`, parses it according to the string `format`, and returns an int that is the number of milliseconds between date and January 1 1970 00:00:00:000 GMT.

The patterns in format are the same as for toDate.

```
fromDate("9-Sep-2001 03:46:40", "d-MMM-yyyy HH:mm:ss") => 1000000000000
```

It is permitted to leave parts out of the date (as we already have seen from the "Date" and "Time" translators.

What will hold is for a format `format` and a string `when`

```
        toDate(fromDate(when, format), format) == when
```

2.9.8.4. various

format

```
        function format(number, fmt, precision)
```

This function returns a string containing the value of `number` formatted according to `fmt` and `precision`.

fmt is a string that field length, sign specification and number format.

Field length is an integral number that specifies the number of characters to be used. The resulting string will be at least as long as the field length. If field length is negative, the string will be right padded with spaces, for a positive field length padding will be left. If field length is omitted, no padding is performed.

The sign specification is either nothing or a plus sign. The plus sign forces a plus to be present on positive numbers.

the number format is one of

d	number is treated as decimal integer.
o	number is treated as octal integer
x	number is treated as hexadecimal integer with result in lower case
X	number is treated as hexadecimal integer with result in upper case
e, E	number is treated as floating point with precision digits after the decimal point. If the number of digits before the decimal point exceeds 14, the format is treated as f or F.
f, F	number is treated as floating point in scientific notation. The number has precision digits after the decimal point (and always one before the decimal point)
g, G	translates to e or E if the exponent is less than precision, to f or F otherwise.

For the decimal formats, the precision is discarded.

For the floating point formats, the difference between the lower - and upper case format letter is shown in a lower - or upper case E for the exponent.

If precision is negative, trailing zero's are omitted from the fractional part. If precision is 0, no decimal point is present.

Some examples

```
        format(1.2345432325252236E22, "f", 3) => "123454323252522.359e8"
        format(1.23, "-8f", -3) => "1.23    "
        format(1.23, "8f", -3) => "    1.23"
        format(1.23, "8+f", 3) => "  +1.230"
        format(123, "x", 0) => "7b"
        format(123, "d", 0) => "123"
        format(123, "o", 0) => "173"
        format(123456.789, "e", -3) => "1.235e5"
        format(123456.789, "f", -3) => "123456.789"
        format(123456.789, "g", -3) => "1.235e5"
        format(123456.789, "g", -6) => "123456.789"
```

precision

```
function precision()
```

This function returns the precision as set in the Settings-tab or the Data Definition Preferences. You can use it as parameter to the format function, or when you want to write your own formatting procedure.

parameter

function parameter(which)

This function returns the current value of a DataThief parameter. `which` is a string, and may contain one of

"look back"

"box size"

"look ahead"

"direction"

"tolerance"

"center"

"imagefile"

The first six return the integer value of the named parameter from the trace controls. The last returns a string with the path name of the current image file. For example

```
parameter("Tolerance")
```

could return 3.

setRuntime

```
function setRuntime(seconds)
```

To prevent DataThief from running for ever when DTC enters an infinite loop, there is a limit to the time DTC will spend in a function. The time is default 5 seconds. When this is not enough, you can change it using this function. The function returns the previous run time limit. Note, that counting run time seconds starts from zero every time you call this function. So if you happen to call this function after 4 seconds with a parameter of 5, the total allowed run time will be 9 seconds.

Note, that when the run time is expired, an error condition is raised. You may catch this error. But as the run time is still expired, you will receive a new run time exceeded error after one second and so on.

message

```
function message(text)
```

This function enters text into text field of the message tab, and opens the message tab. You can use it to alert the user of an error condition. The message will remain visible as long as the code that calls the message function is used. Suppose you have a translator that requires a positive number, then you will have the following code

```
function translate(v)
{
   local value = eval(v);
   // no need to catch errors, those will give a message anyway
   if(value <= 0)
   {
      message("Translator requires positive number, found " + v);
      return;
   }
   ....
}
```

As soon as all coordinate values result in positive values, the message tab will hide itself.

eval

```
function eval(text)
```

This function treats text as if it was DTC expression and returns the result. This can, as in the above example, be used to convert a string of digits to a numeric value, but you can also use it to create a new expression as in

```
a = '121";
b = "14";
c = eval(a + "+" + b);
d = a + b;
e = eval("sqrt(" + a + ")")
```

The result is of course that `c` contains 135, `d` contains "12312" and e contains 11.0;

2.9.9. A full scale example

For a complete example of DTC in DataThief we shall write our own equivalent of the date translator that we preciously implemented using toDate() and fromDate().

What we need is a function translating a stringin the form "day-month" to a number and a second function translating a number back to a string in the from "day-month".

Let us start with the first. Obviously, we need to know the names of the months, so we define

```
mname = [ "jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep",
"oct", "nov", "dec" ];
```

And we need to know howmany days each month has - disregarding leap years.

```
mlength = [ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 ];
```

Now we define the basis of the the function to make a number out of a date. In this first draft we do not yet check for errors; we assume that the input is valid.

```
function day2num(date)
{
      // date is day-month, split it on the hyphen
      local parts = split(date, "-");
      // find out which day of the month we have
      // it is still a string, so eval() to convert to int
      local days = eval(parts[0]);
      // add days from previous months
      for(local month = 0; month < sizeof mname; month++)
      {
         if(mname[month] == parts[1])
         {
            // we have found the requested month, return result
            return days;
         }
         days += mlength[month];
      }
      // hey, we have not found the month, this should be an error...
      message("No month found in " + date);
}
```

This function returns 1 for "1-jan" and 365 for "31-dec". We show a message when the month is not found, but input like "a-jan" wil generate an error "a not defined" (in the `eval(parts[0])`). Also, because string comparison is case sensitive, "1-Jan" will not give a correct result. To address the last problem first, we define a new function that changes every upper case character in a string to lower case. The change is made in the string itself, but the result is returned nevertheless.

```
function lowerCase(s)
{
   for(local i = 0; i < sizeof(s); i++)
   {
      if(s[i] >= 'A' && s[i] <= 'Z')
      {
         s[i] += 'a' - 'A';
      }
   }
```

```
        return s;
    }
```
And now we do the function day2num again
```
    function day2num(date)
    {
        local parts = split(date, "-");
        if(sizeof parts != 2)
        {
            // no hyphen in the input
            message("Can not make a date out of " + date);
            return;
        }
        // find out which day of the month we have
        local days = eval(parts[0]);
        if(typeof days != "int")
        {
            message("Invalid day in " + date);
            return;
        }
        // make month lower case
        lowerCase(parts[1]);
        // add days from previous months
        for(local month = 0; month < sizeof mname; month++)
        {
            if(mname[month] == parts[1])
            {
                // we have found the requested month, return result
                return days;
            }
            days += mlength[month];
        }
        message("No month found in " + date);
    }
```
Now we define the function to create a date out of a number
```
    function num2day(num)
    {
        // first check validity of input
        if(num < 1)
        {
            num = 1;
        }
        if(num > 365)
        {
            num = 365;
        }
        local month = 0;
        while(num > mlength[month])
        {
            num -= mlength[month++];
        }
        // we are at the month, but we don't want "12.4-feb"
        return round(num) + "-" + mname[month];
    }
```
The file date.dtc, that can be downloaded from http://www.datathief.org, contains this code. You can of course type it into the Function Preferences panel, but you can open DTC files, provided the file has the extension .dtc.

Once you have the functions defined, create a new Translator "date2" with

Trans = `num2day(v)`

Reverse = `day2num(v)`

And select this translator in your output definition.

When we want to specify a distance in the output system for data point culling, we can of course redefine day as

```
        day = day2num("2-feb") - day2num("1-feb")
```

but we know that with this translator day is 1. (as we know now that in the older date translator day was 24 * 3600 * 1000 (the number of milliseconds in one day)).

2.10. Known problems

Software hardly ever is entirely without problems. When a program is supposed to run on platforms as different as Linux, Windows and MacOS the number of problems probably only increases.

This version of DataThief has been developed on a system running Fedora Core 2 and J2ee SDK 1.4. It has extensively been tested on this platform and on a system running Windows XP SP2 using J2se RE 5.0.1. Further test have been done on a Macintosh running MacOS 8.6 and MRJ 2.2.5. and another Macintosh running MacOS X.3. A testrun has been performed on a system running SUN Solaris. The basic functionality on all these systems was OK.

Actually the only problems I have found until now - I regret to say - is on the development system.

Linux

DataThief stuck after window close

On my Linux system using J2ee SDK 1.4, it happens that the Java Interpreter seems to stop processing events for some time (typically several seconds) after closing a window, both dialogs and the main window. Eventually processing resumes, so if this happens, please exercise patience.

Mouse wheel does not work properly

On a Linux system using J2ee SDK 1.4, the mouse wheel alters the position of the vertical scroll bar, but this does not affect the view area.

If you encounter problems using DataThief please mail them to

bas@datathief.org