Description

Self-test of the Imagefitter `tool`. This function is very limited presently as it's hard to test an interactive application.
To run a specific test, you can use the `which` argument. Since the number of tests keeps growing, we don't tell you how many there are here or what exactly they do ! If you give too large a value, nothing will happen. `which` can be an integer or a vector of integers.

Example

```
- imagefittertest()
```

Runs all tests.

1.2.16 imageprofilefitter - Tool

Package general
Module images

Fit models to 1D profiles from images
include "imageprofilefitter.g"

Constructors

imageprofilefitter	Construct an imageprofilefitter

Functions

done	Destroy this imageprofilefitter tool
getestimate	Recover the estimate
getfit	Recover the fit
getprofiles	Get data, estimate and fit as vectors
getstore	Recover all fits added to internal store
gui	Start the GUI interface
setimage	Set a new image
type	What type is this tool ?

The imageprofilefitter `tool` offers interactive fitting of 1D models (or components) to profiles from an image. You control the process from a GUI interface. The `tool` is self contained so that the display is built into the interface.
You may provide one or two images (primary in keyword `infile` and

secondary in keyword `infile2`). If you only provide the primary image, it must have at least 3 dimensions. Then, a zeroth-order moment secondary image (simple average of all pixels along profile with 3-sigma clip) is automatically made for you along the specified profile axis (and removed when you destory the Imageprofilefitter tool or exit from Glish). This secondary image is then displayed and from it you interactively specify profiles to fit. If you provide the secondary image as well, it could be a pre-computed moment image (you might not like the automatically provided one). The secondary image does not actually have to be generated from the primary It is only its world coordinates that are used. For example, it could be an optical image from which you specify a direction and the profile is then extracted from the primary image (which might be a spectral-line neutral Hydrogen cube).

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube', axis=3)
```

Specify just the primary image.

Example

```
- pf2 := imageprofilefitter(infile='ngc1313.cube', infile2='ngc1313.mom', axis=3)
Specify the primary and secondary images.
```

The display of the secondary image looks like:
There are now two ways to drive the fitting process. You can either interactively fit individual profiles one at a time, or you can automatically fit all profiles in a selected region.

Individual Profiles
From the Viewer control buttons (buttons to the left of the display of the image), select the position handling button. Now go to the image display, position the cursor somewhere and double click. This will cause a second GUI to appear as below.
The profile is extracted from the nearest pixel and plotted in the secondary GUI. If you wish, the profile can be spatially averaged, rather than just being a single-pixel profile. Just enter the half-width (in pixels) in the entry box of the main GUI (defaults to 0; no averaging). The title of the plot includes the world and center pixel coordinate of the profile.
A profile can also be generated by selecting one of the region generation tools (rectangle or polygon) from the Viewer control buttons. If the Region Usage menu has 'Average' selected, then the profile will be spatially averaged over the region. Generate the region on the image display by dragging out the box or marking the polygon. As always, double click within the region to emit it.

There are various buttons and menus beneath the plot of the profile. You can select different x-axis units, absolute or relative units etc. From here the process is simple. You select the number of components you want to fit with the slider. For each component, a parameters panel is generated (you access different components with the TAB-widget buttons). Then for each component you specify the type of component you wish to fit. For each component you need an estimate before fitting.

You can either do this interactively or automatically. To make an interactive estimate, select the component from the TAB-widget. Then press the 'Inter' button and follow the instructions in the message widget at the bottom of the GUI.

To make an automatic estimate, just press the 'Estimate' button at the bottom of the GUI. It will make an estimate for as many components as you have specified (i.e. you don't do it for each component separately).

The estimate is plotted for you. Then you can press the 'Fit profile' button and the fit will be done. The fit and residual will be plotted.

If you don't want to see all of the plots (data, estimate, fit, residual), you can select what you want from the 'File' menu of the menubar at the top of the GUI.

That's it ! If you want to fix parameters of the fit at the estimate value, check the 'Fix' button for the appropriate component parameter. You can transfer the fit solution to the estimate for each parameter by pressing the left-pointing arrow.

The results are presently stored in a *Glish* record. You can copy it to the **defaultclipboard** `tool` by pressing the 'Copy' button at the bottom the GUI. You can then access the results from *Glish* by pasting from the clipboard.

```
- r := dcb.paste()      # Paste from clipboard to record
- print field_names(r)
estimate fit position
- print field_names(r.fit)
elements xunit xabs yunit doppler nmax baseline
```

Presently, the results are held in a *Glish* record with fields 'estimate', 'fit' and 'position' (the structure of this record is **very** likely to change in the future). There are a variety of subfields with obvious names. To access the results of the fit you would look at the `fit.elements` field of the record (above). This record is indexed by the number of components (although *Glish* record indexing is rather perversely named; you see funny field names like *445, *446 - ignore these). What you would do to see the individual components, assuming say, you fitted two components to your spectrum would be

```
- print r.fit.elements[1]
[type=GAUSSIAN, parameters=[0.60423172 1.4136544 0.00022220827] ,
               errors=[0.042635564 7.6873681e-06 1.8104964e-05] ]
- print r.fit.elements[2]
```

```
[type=GAUSSIAN, parameters=[0.13442206 1.4142888 4.6772192e-05] ,
                errors=[0.093013123 1.588148e-05 3.7512798e-05] ]
```

You can also add the result from each fit to an internal store via the 'Store' button at the bottom of the GUI. The internal store of all the fits can be recovered with the function getstore.

Fitting many profiles in a region
This mode is active when you have selected the Region Usage menu to say 'Fit all profiles'.
From the Viewer control buttons (buttons to the left of the display of the image), select one of the region handling buttons (box or polygon). Now go to the image display, and generate the region by dragging out the box or marking the polygon. As always, double click within the region to emit it. This will cause a second GUI to appear. Is is the same GUI as described above for fitting individual profiles.
The spectrum that is plotted, is the spectrum at the bottom-left corner of your region. You use this GUI to indicate how many components you want to fit, what parameters are fixed and if you wish, and to set an initial estimate for the first profile. Thereafter, an automatic (independent) estimate is made for each profile.
When you press the 'Fit region' button, all profiles in the delineated region are automatically fit. Two new output images are created for you. They have the same name as the input image but with `.fit` and `.resid` appended to them. These images hold the evaluated fit and resdidual.

Non-interactve profile selection
Imageprofilefitter also has a mode where the image is not displayed (`showimage=F` in constructor). This means you cannot interactively select the spatial location at which the next profile will be extracted.
What happens then is that the current image (set in the constructor or via function setimage) is used to generate a profile averaged over the entire image (for the non-profile axes). It is used in this way in Dish; each single-dish spectrum is put into a 1-D image and passed to Imageprofilefitter.

Frame conversions
At the moment, when you plot profiles, you will find that spectral frame conversions are disabled because that functionality is not yet implemented in the fitter (although the plotter can handle it). If you really need to fit in a different reference frame, you can do something like the following:

Example

```
- im := image('ngc1313.cube')
- cs := im.coordsys()
- cs.setconversiontype (spectral='TOPO')       # Convert from native frame to TOPO
```

```
- im.setcoordsys(cs)
- cs.done()
- im.done()
- p := imageprofilefitter('ngc1313.cube')
```

and now the plotting and fitting will be done in the TOPOcentric frame.

Future Enhancements
This is the first version of this `tool`. Future improvements will include

- Enable and handle reference frame changes (e.g. LSRK to BARY etc)
- Line shapes other than Gaussian (e.g. Zeeman, Voigt)
- Line-ratio constraints
- Interactive capability when fitting many profiles in the region
- Robust estimates of line parameters
- Zooming in and out of the profile display and setting ranges

imageprofilefitter - Constructor

Package	**general**
Module	**images**
Tool	**imageprofilefitter**

Construct an imageprofilefitter

Synopsis
imageprofilefitter(infile, infile2, sigma, parent, axis, plotter, showimage, widgetset)

Arguments

infile	in	The primary image file name or image tool Allowed:String or Image tool
infile2	in	The secondary image file name or image tool Allowed:String or Image tool Default: Auto generate
sigma	in	Weights image Allowed:String Default: None
parent	in	Parent frame to embed GUI in Allowed:Tk frame agent Default: F
axis	in	Profile pixel axis Allowed:Integer Default: Spectral
plotter	in	The plotter for profile plots Allowed:unset or a pgplotter tool Default: Will create its own
showimage	in	Display the image ? Allowed:T or F Default: T
widgetset	in	The widgetset to use for the GUI Allowed:A widgetserver tool Default: ddlws

Returns
imageprofilefitter tool

Description

This constructor creates the primary GUI. You must specify the primary image via the `infile` argument. The secondary image specified by `infile2` is optional. If you give it, this is the image that is displayed from which you indicate which profiles to fit. If you don't give it, it is generated automatically from the primary image by averaging the data over the profile axis with a $3-\sigma$ clip applied. You can specify the input images either by their file names or by giving an `image tool`.
You may specify a fitting weights image via argument `sigma`. It must have the same shape as the primary image. If you don't give it, all pixels have the same weight in the fitting process.
The profile axis is specified by the `axis` argument. If you leave it unset, then the spectral axis, if there is one, is selected. Otherwise axis 3 is selected.
You may optionally embed the display in an existing Tk frame (`parent`) if you wish.

By default, the Pgplotter `tool` that is used to display the profiles is created for you. But you may wish to pass in your own plotter (perhaps it's embedded in some other GUI). If so, use the `plotter` argument to pass in a Pgplotter `tool`. Finally, `showimage`, when set to F, instructs the Imageprofilefitter not to display the image. This means you cannot select profiles interactively. What happens then is that the current image (set in the constructor or via function setimage) is used to generate a profile averaged over the entire image (for the non-profile axes).

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube', infile2='locate.image', axis=3)
- pf.done()
```

---

imageprofilefitter.getestimate - Function

Package	**general**
Module	**images**
Tool	**imageprofilefitter**

Recover the estimate

Synopsis
getestimate()

Returns
Glish record

Description

When you are fitting individual profiles, this returns the current estimate into a record. If you are fitting many profiles in a region this function returns an empty record.
Field 'elements' holds the parameters of each component. Thus the length of field 'elements' is the number of components you fit. Its subfields are self-explanatory when you look at it.
Field 'xunit' gives the abcissa unit. Field 'xabs' says whether the x-values are absolute or relative (to reference pixel). Field 'yunit' gives the ordinate unit. Field 'doppler' gives the doppler type if the profile axis is spectral.

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube')
- est := pf.getestimate()
```

---

imageprofilefitter.getfit - Function

Package	**general**
Module	**images**
Tool	**imageprofilefitter**

Recover the fit

Synopsis
getfit()

Returns
Glish record

Description

This returns the current fit into a record.
If you are fitting individual profiles then the returned record has two fields, 'fit' and 'position'. The 'fit' field holds all the fields described in function getestimate. The 'position' field holds the position of the profile in a variety of formats (pixel, world, string formatted etc). It's self explanatory on inspection.
If you are fitting many profiles in a region then the length of this record is the number of profiles you fit in the region. Each field holds subfields 'position' and 'fit'. The 'fit' subfield has all the subfields described in getestimate.

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube')
- fit := pf.getfit()
```

---

imageprofilefitter.getprofiles - Function

Package	**general**
Module	**images**
Tool	**imageprofilefitter**

Get data, estimate and fit as vectors

Synopsis
getprofiles(outfile, ascii, format, dataonly, overwrite, ack)

Arguments

outfile	in	Output file Allowed:String Default: None
ascii	in	Save in ascii file or aips++ table ? Allowed:Boolean Default: F
format	in	Formatting string for ascii table Allowed:String Default: %e
dataonly	in	Only write out data, not estimates and/or fits Allowed:Boolean Default: F
overwrite	in	Overwrite output file if it exists ? Allowed:Boolean Default: T
ack	in	Acknowledge Table creation/writing to logger ? Allowed:Boolean Default: F

Returns
Record or T

Description

This function recovers vectors holding the abcissa, data, estimate, fit, residual (these three are optional) and mask for the interactive fit just done. The abcissa is saved in both absolute pixel and world coordinates. For the latter, it will be in the units (and doppler if relevant) as displayed on the plotter.
If you don't specify an output file name, these vectors are returned in a record to you with field names as above.
If you specify an output file name you can save the data as an aips++ Table (default) or an ascii file. For the latter you can specify the floating point format. For the latter the mask will be stored as 0 (bad) or 1 (good).
If you store as an AIPS++ table, the units are stored as column keywords. If you store as an ascii table, the units are written as the first line. If the abcissa

units are consistent with velocity, then the doppler type is written out as well. So you will see on the first line, 'xunit yunit' or 'xunit doppler yunit'. The second line gives the column headers.

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube', infile2='locate.image', axis=3)
-                                     # User does a fit
- r := pf.getprofiles();              # Recover into record
- field_names(r);
abcissa data estimate fit residual mask
-
- pf.getprofiles(outfile='junk.tbl')  # Save as aips++ table
- t := table('junk.tbl')              # Open
- t.browse();                         # Browse
```

---

imageprofilefitter.getstore - Function

Package	**general**
Module	**images**
Tool	**imageprofilefitter**

Recover all fits added to internal store

Synopsis
getstore()

Returns
Glish record

Description

This returns all of the fits that you have added to the internal store (by pressing the 'Store' button) into a record.
There is one (numerically labelled) field in the record for each press of the 'Store' button.
Each of these records has further fields 'estimate', 'fit', and 'position' holding the obvious things.

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube')

# do some fits

- s := pf.getstore();
- print 'Number of components = ', length(s)
Number of components = 10
-
- field_names(s[5])
estimate fit position
```

---

imageprofilefitter.done - Function

Package	**general**
Module	**images**
Tool	**imageprofilefitter**

Destroy this imageprofilefitter tool

Synopsis
done()

Returns
Bool or fail

Description
This function destroys the Imageprofilefitter `tool`.

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube')
- pf.done()
- pf
F
```

---

imageprofilefitter.gui - Function

Package general
Module images
Tool imageprofilefitter

Start the GUI interface

Synopsis
gui()

Returns
Bool or fail

Description
If you dismiss the GUI (with the dismiss button), then you can recover it with this function. The state of the Imageprofilefitter and the GUI is preserved. If the GUI is already present, then it does nothing.

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube')
- pf.gui()
```

---

imageprofilefitter.setimage - Function

Package general
Module images
Tool imageprofilefitter

Set a new image

Synopsis
setimage(infile, infile2, sigma)

Arguments

infile	in	The primary image file name or image tool Allowed:String or Image tool
infile2	in	The secondary image file name or image tool Allowed:String or Image tool Default: Auto generate
sigma	in	Weights image Allowed:String Default: None

Returns
Bool or fail

Description
You can set a new image (primary and secondary) with this function. Doing so will cause the displays to be updated.
You may specify a fitting weights image via argument `sigma`. It must have the same shape as the primary image. If you don't give it, all pixels have the same weight in the fitting process.

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube')
- pf.setimage(infile='sgra.cube')
```

---

imageprofilefitter.type - Function

Package	**general**
Module	**images**
Tool	**imageprofilefitter**

What type is this tool ?

Synopsis
type()

Returns
String or fail