1.8.9 typestest - Function

Package **utility**
Module **tasking**

Test the types definition

Synopsis
typestest()

Description
This defines a class called typestest, a global function called typestester, and the arguments for typestester. The toolmanager is then called to display the interface for typestester, which may then be executed to see the arguments passed in.

1.8.10 plugins - Tool

Package **utility**
Module **tasking**

attach code to tools at runtime
include "plugins.g"

Constructors

Functions

attach	Attach plugins for a type to an tool
loaded	list which plugins have been loaded
loadplugins	load plugin (.gp) files for some class
rescan	Find all plugin (.gp) files

Description
`plugins` is the technique by which additional Glish functions are attached to tools when that tool is created. For example, a user might want to attach some code that automatically fits models to data to a pgplotter, or a site might wish to attach telescope-specific calibration functions to standard data reduction classes.

The function is a simple one. When `plugins.g` is first included, it searches all directories in its include path (Glish variable `system.path.include`) for files of the form `class_*.gp`, where `.gp` means "Glish plugin", "class" is the tool type that the plug refers to, and the "*" is a category for the types of functionality within that plugin. For example, `pgplotter_fitting.gp` might be the name of a plugin for attaching fitting functionality to a pgplotter.
The files are not actually loaded at this stage, they are just located. Then when an tool of a type that has plugins is actually created, the plugin code is attached to the tool just before the tool is returned to the user.
The plugin system is careful to do as little work as possible. In particular, it loads files only on demand, and does not require any services from the rest of AIPS++that might take more than a brief time to start.
`plugins` is a single tool. Not more than one such tool can ever be created (automatically via the inclusion of `plugins.g`) in any Glish session.

Example
If you want to plugin-enable your classes, you merely have to:

1. include `plugins.g` at the beginning of your `.g` file; and
2. include Call `plugins.attach` before returning the tool from your constructor

That's it! It is so simple and low-cost, that generally speaking all user-level classes should be plugin-enabled.
Here is a simple example.of a plugin-enabled tool that just keeps track of a counter:

```
pragma include once
include 'plugins.g'                          ### 1

counter := function() { # constructor
    private := public := [=];
    private.count := 0;

    # Member functions
    public.bump := function() {
        wider private;
        private.count +:= 1;
        return T;
    }

    public.total := function() {
        wider private;
        return private.count;
    }
```

```
    plugins.attach('counter', public);      ### 2
    return ref public; # tool return
}
```

The numbered lines are all that is necessary to plugin-enable the counter class. The unaugmented class can be used like this:

```
- c := counter()
- c.bump(); c.total();
T
1
```

Now, suppose we are a user who wants to customize this simple counter so that we can "bump" the counter an arbitrary number of times. We could create the following plugin to attach a "multibump" function.

```
# counter_mychanges.gp
pragma include once;

counter_mychanges := [=];

counter_mychanges.attach := function(ref public) {
    # Multibump function
    public.multibump := function(count) {
        for (i in 1:count) { public.bump(); }
        return T;
    }

    return T; # Say that the attach worked
}
```

Note how we can both extend the list of functions in the plugin by just adding a function to public, and how we can use other public functions. There is no access to the private state of the class through a plugin – it preserves encapsulation.

A careful programmer would check to see whether or not a "multibump" function already exists before adding one, and returning a "fail" (probably via the throw function) if it does.

Now we can just use the multibump function as if it was defined in the original class, even though it has been added after the fact:

```
+ c.multibump(5); c.total()
T
6
```

If there was any one-time initialization code that was needed by any of the functions in the file, you would implement them in a function:

```
counter_mychanges.init := function() # no arguments
{
    ...        # initialization code
    return T;  # or fail if there is an error
}
```

plugins.rescan - Function

Package	**utility**
Module	**tasking**
Tool	**plugins**

Find all plugin (.gp) files

Synopsis
rescan()

Description
This function is used to find all plugin files. It is called automatically when the `plugins` tool is constructed, so you generally never need to call it. You might call it if you know that some new `*.gp` files have been created since you started your Glish session.
The rules are that it searches all directories in `system.path.include` for `.gp` files that are of the form `class_something.gp`, where "class" is the classname this plugin is to be attached to, and something is a general description (category) of what functionality is provided by the plugin (for example, "demo").

---

plugins.loadplugins - Function

Package	**utility**
Module	**tasking**
Tool	**plugins**

load plugin (.gp) files for some class

Synopsis
loadplugins(type)

Description

This function loads all the plugin files for some class (*i.e.*, type). You should never have to call it, since `plugins` itself will call it the first time it is needed for that class.
If a plugin has an "init" function, `loadplugins` will call it after the file has been included.

Arguments

type	type (class) for which we want to load plugins Allowed:string

---

plugins.loaded - Function

Package	**utility**
Module	**tasking**
Tool	**plugins**

list which plugins have been loaded

Synopsis
loaded()

Description
This function returns a record with information about what plugins have been loaded. It is probably not of any general interest.

Returns
record

---

plugins.attach - Function

Package	**utility**
Module	**tasking**
Tool	**plugins**

Attach plugins for a type to an tool

Synopsis