function_handle (@)

Handle used in calling functions indirectly

Syntax

```
handle = @functionname
handle = @(arglist)anonymous_function
```

Description

`handle = @functionname` returns a handle to the specified MATLAB function.

A function handle is a MATLAB value that provides a means of calling a function indirectly. You can pass function handles in calls to other functions (often called *function functions*). You can also store function handles in data structures for later use (for example, as Handle Graphics callbacks). A function handle is one of the standard MATLAB data types.

At the time you create a function handle, the function you specify must be on the MATLAB path and in the current scope. This condition does not apply when you evaluate the function handle. You can, for example, execute a subfunction from a separate (out−of−scope) M−file using a function handle as long as the handle was created within the subfunction's M−file (in−scope).

`handle = @(arglist)anonymous_function` constructs an anonymous function and returns a `handle` to that function. The body of the function, to the right of the parentheses, is a single MATLAB statement or command. `arglist` is a comma−separated list of input arguments. Execute the function by calling it by means of the function handle, `handle`.

Remarks

The function handle is a standard MATLAB data type. As such, you can manipulate and operate on function handles in the same manner as on other MATLAB data types. This includes using function handles in structures and cell arrays:

```
S.a = @sin;  S.b = @cos;  S.c = @tan;
C = {@sin, @cos, @tan};
```

However, standard matrices or arrays of function handles are not supported:

```
A = [@sin, @cos, @tan];        % This is not supported
```

For nonoverloaded functions, subfunctions, and private functions, a function handle references just the one function specified in the `@functionname` syntax. When you evaluate an overloaded function by means of its handle, the arguments the handle is evaluated with determine the actual function that MATLAB dispatches to.

Use `isa(h, 'function_handle')` to see if variable h is a function handle.

Examples

Example 1 −− Constructing a Handle to a Named Function

The following example creates a function handle for the`humps` function and assigns it to the variable `fhandle`.

```
fhandle = @humps;
```

Pass the handle to another function in the same way you would pass any argument. This example passes the function handle just created to`fminbnd`, which then minimizes over the interval `[0.3, 1]`.

```
x = fminbnd(fhandle, 0.3, 1)
x =
    0.6370
```

The `fminbnd` function evaluates the `@humps` function handle. A small portion of the `fminbnd` M−file is shown below. In line 1, the `funfcn` input parameter receives the function handle `@humps` that was passed in. The statement, in line 113, evaluates the handle.

```
1     function [xf,fval,exitflag,output] = ...
          fminbnd(funfcn,ax,bx,options,varargin)
                .
                .
                .
113   fx = funfcn(x,varargin{:});
```

Example 2 −− Constructing a Handle to an Anonymous Function

The statement below creates an anonymous function that finds the square of a number. When you call this function, MATLAB assigns the value you pass in to variable `x`, and then uses `x` in the equation `x.^2`:

```
sqr = @(x) x.^2;
```

The `@` operator constructs a function handle for this function, and assigns the handle to the output variable `sqr`. As with any function handle, you execute the function associated with it by specifying the variable that contains the handle, followed by a comma−separated argument list in parentheses. The syntax is

```
fhandle(arg1, arg2, ..., argN)
```

To execute the `sqr` function defined above, type

```
a = sqr(5)
a =
   25
```

Because `sqr` is a function handle, you can pass it in an argument list to other functions. The code shown here passes the `sqr` anonymous function to the MATLAB quad function to compute its integral from zero to one:

```
quad(sqr, 0, 1)
ans =
   0.3333
```

See Also

`str2func`, `func2str`, `functions`, `isa`