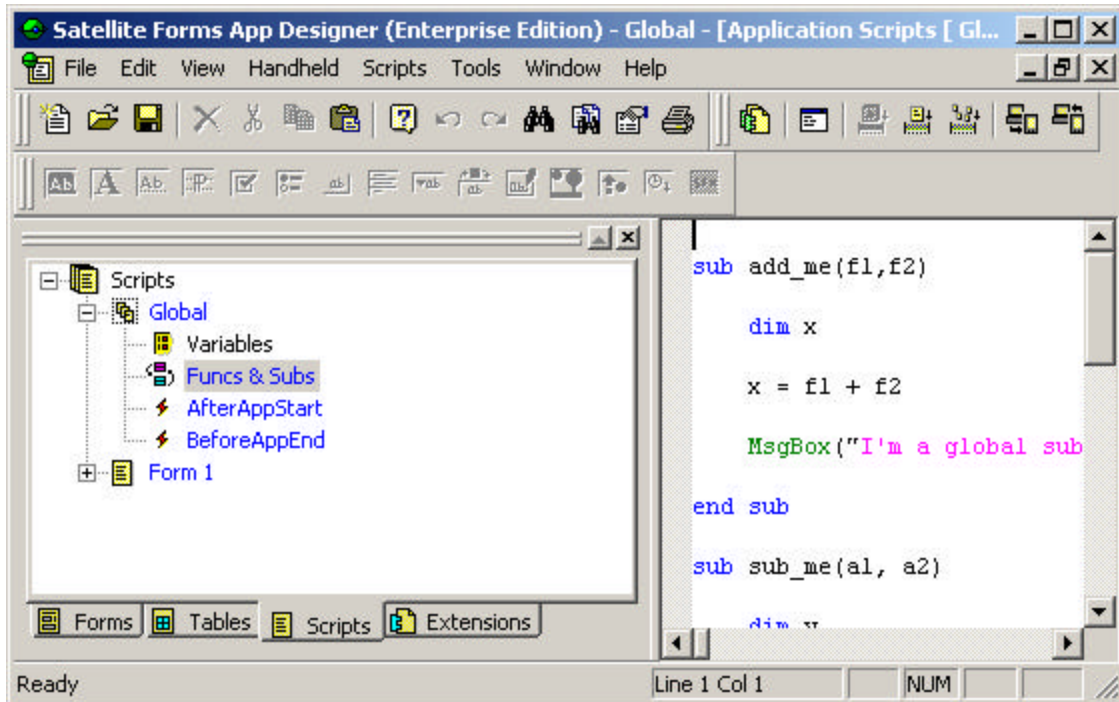


**From:** PUMATECH, Inc  
**Created:** Tuesday, August 15, 2000  
**Subject:** Creating Subroutines and Functions with Satellite Forms 4.0 EE and SE

This doc covers how to use the new subroutine feature of Satellite Forms 4.0 EE and SE.

Subroutines and functions must be written in the "Funcs & Subs" area of the Scripts tree-view (see picture below).



Routines defined in this window will be accessible from any script within the SF application. Like Visual Basic, Satellite Forms supports 2 varieties of routines: function and subroutine.

Functions must return result values; whereas, subroutines do not return anything.

Subroutines and functions can be called from any event in Satellite Forms.

#### Note

While the syntax below looks similar to Visual Basic, Satellite Forms only implement a subset of the features offered by Visual Basic.

#### Notation in the Syntax declarations below

- **Bold** notation denotes keywords
- Items within a pair of [] square brackets are optional
- *Italics* notation denotes user defined parts

#### 1. Sub Statement

##### Syntax

**Sub** *name* [(*arglist*)]

```
[statements]
[exit]
[statements]
End Sub
```

#### Parts

*name* name of the subroutine which follows standard SF variable naming conventions

*arglist* list of variables that must be passed to the **Sub** separated by commas

*statements* valid SF scripts to be executed within the **Sub**

#### **Remarks**

Local variables can be defined using the **Dim** keyword. The value of local variables in a **Sub** is not preserved between calls to the routine.

You cannot define a Sub within a Sub or a Function.

Calling a Sub is as simple as writing its name and its variable. Unlike Visual Basic, there is no need to use the keyword **Call**.

Unlike Visual Basic, the **Exit** keyword cannot be fully qualified (i.e. in Visual Basic, you would have to write **Exit Sub** to exit the routine).

Functions and Subs can call themselves repeatedly (recursive). Be careful however as excessive recursion can lead to stack overflow.

## **2. Function Statement**

#### **Syntax**

```
Function name [(arglist)]
    [statements]
    [name = expression]
    [exit]
    [statements]
    [name = expression]
End Function
```

#### Parts

*name* name of the subroutine which follows standard SF variable naming conventions

*arglist* list of variables that must be passed to the **Function** separated by commas

*statements* valid SF scripts to be executed within the **Function**

*expression* return value of the **Function**

#### **Remarks**

Local variables can be defined using the **Dim** keyword. The value of local variables in a **Function** is not preserved between calls to the routine.

You cannot define a Sub within a Sub or a Function.

Calling a Sub is as simple as writing its name and its variable. Unlike Visual Basic, there is no need to use the keyword **Call**.

Unlike Visual Basic, the **Exit** keyword cannot be fully qualified (i.e. in Visual Basic, you would have to write **Exit Function** to exit the routine).

Functions and Subs can call themselves repeatedly (recursive). Be careful however as excessive recursion can lead to stack overflow.

### Example

Below is an example of a function that recursively computes the factorial of a positive integer:

**Function** factorial ( num )

```
    if (num <= 0) then
        MsgBox("Can only take factorials of positive numbers")
        Exit
    endif

    if (num = 1) then
        factorial = 1
    else
        factorial = num * factorial( num - 1 )
    endif
```

**End Function**

Below is an example of using the factorial example above from within a script:

```
Dim result
result = factorial( 6 )
MsgBox( "Factorial of 6 is " & result)
```