

| | | |
|-----------|--------|-------------------------|
| procedure | String | Procedure to be called |
| stack | Number | Stack size in bytes |
| process | String | Name of process created |

Creates a new process using the string Procedure and returns its ID. The process is given a stack size of Size. The optional parameter Name specifies the name of the process. If Name starts with '\$', the process is local under 4D Server.

New process launches a new process with the name process, and returns a reference to the process.

Procedure is a string specifying the name of the procedure that should be executed in the process.

Stack is the amount of memory allocated for the stack of the process. It is the space in memory used to store process objects like procedure calls, local variables, parameters in subroutines, stacked records, and recursive routines. It is expressed in bytes and depends on the number of objects created by the process. A range of 16000 to 32000 is adequate for most processes. The minimum stack size is 16000.

The optional parameter process is the name of the process. Process is used to identify the process in the Process List editor in the Design environment. If you do not specify the name, your process will be called "Process_xx".

4D SERVER: In the case of the 4D Server the name is displayed in the main window for that particular user. A local process may be created by prefixing the name of the process with the '\$' character. This allows the 4D Server to continue processing operations that are server intensive. A local process can be used when you do not need to access data.

When a process is launched, the process has its own menu bar. By default this is a blank menu bar. Use the command MENU AR to set the menu bar for a process. The process can have one or more windows open at the same time, but a process can only have one frontmost window.

Each process can have its own current selection and current record for each file. When a process is launched, the current selection for each file in the new process is empty and there is no current record. Every process has its own process variables.

There is no default file for a process until the command DEFAULT ILE is executed for each process. Each process can then have its own default file. For each file, each process can have different input and output layouts. The default input and output layouts for a file can be specified in the layout dialog box in the Design Environment.

Each process can have its own process sets. Every process has its own UserSet and LockSet. Every process can have its own debugger. To display the debugger for a process use the RACE command or select the process in the Process List Editor and choose Trace from the Process Menu.

The number of processes that can be created are limited by memory. With 4th Dimension, this is dependant on the memory of the machine. With 4D Server, this is dependant on server memory. If a process could not be created the New process function returns 0.

In the following example a local process called '\$ProcessName' is created. The stack size is 32000 bytes and the procedure that controls the process is 'myProcedure'.

```
vProcID:=New process("myProcedure";32000;"$ProcessName")
```

The following example creates two separate processes. The first process called CustList launches the ShowCust procedure as a separate process. The CustList procedure displays all the records in the [Customers] file. The VendorList process displays the list of vendors using the same method:

```
C_LONGINT (◊ModCust; ◊ModVendors)
◊ModCust := New process ("ShowCust"; 32000; "CustList")
◊ModVendors := New process ("ShowVend"; 32000; "VendorList")
```

The procedure ShowCust displays the records in the file [Customers]:

```
ALL RECORDS ([Customers])           ` Select all the records
MODIFY SELECTION ([Customers];*)    ` Show the selection
```

The following example launches four processes consecutively. The second process is not launched until the first process initializes the interprocess variable ◊Launched to true. Since the first process initializes ◊Launched to true, this stops the second process from being created while the first one is still in the process of being created. In the same way, the third process will not be created while the second process is being created and so on. To implement this distributed method of creating processes, we first create an array that will store the process reference numbers as the processes are created:

```
C_BOOLEAN(◊Launched)
ARRAY LONGINT(◊ProcArray;4)    ` Initialize a four element array

For ($i;1 ;4) ` Loop through and store each process
    ` references in each element of the array
    ◊ProcArray$ i:=New process("Procedure"+String($i);32000)
    Repeat
        IDLE        ` for compiled databases
        ` This forces each process to wait for its predecessor to
        ` be launched before continuing. The procedures that get
        ` launched set the variable to true.
    Until (◊Launched)
    ◊Launched:=False    ` Reset for the next process...
End for
```

See also: ALL RECORDS, Frontmost process,
MODIFY SELECTION