

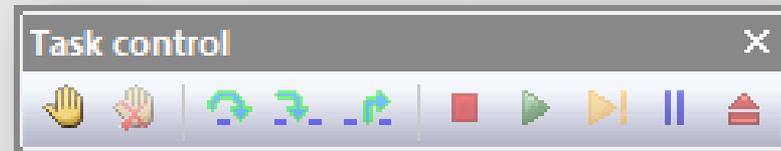
softMC Training – Module 5

# Running, Controlling and Debugging a Task



# Contents

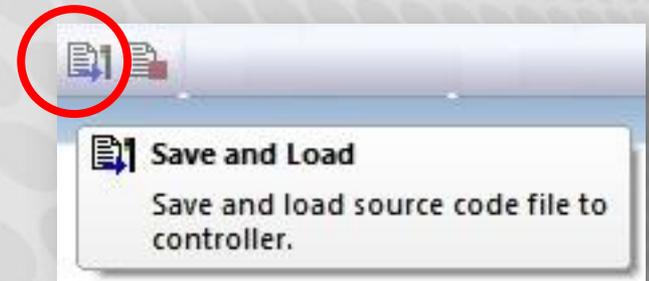
- Load, Run, Idle, Pause, Abort
- Multiple instances of a task
- Multitasking – multiple tasks running simultaneously
- Priority
- Task status
- Debugging



# Loading and Starting a Task

# Loading a Task

- **Load** command loads a task/library from the flash memory into softMC **RAM**
  - from Terminal
  - from another task
  - from autoexec.prg
- Syntax/Example
  - Load `MyTask.prg`
- Program/library must be loaded to RAM in order to be executed
- Syntax is checked when the task is loaded
- Syntax errors are written to the TRN.ERR file
- Program with syntax error will not be executed
- ControlStudio
  - Use the **Save and Load** button



# Running a Task

- **StartTask** command starts execution of a task
  - from Terminal
  - from another task
  - from autoexec.prg

- Syntax

```
StartTask <task> {Priority = <level>} {NumberOfLoops = <number>}
```

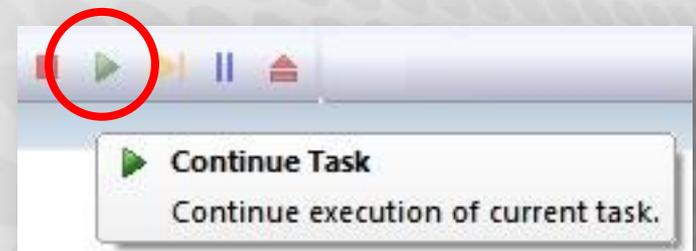
- Example

```
StartTask MyTask.prg Priority = 6
```

```
StartTask MyTask.prg NumberOfLoops = 5
```

- ControlStudio

- Use the **Continue Task (Run)** button



# Idling a Running Task

- **IdleTask** command stops the task at the end of the line currently being executed and idle all its events
- Syntax/Example  
`IdleTask MyTask.prg`
- IdleTask does not stop motion currently being executed
- An idled task can be continued (ContinueTask) or terminated (KillTask)
- ControlStudio
  - Use the **Idle Task** button



# Pause Task

- **PauseTask** command idles the active program at the next Pause command within the program

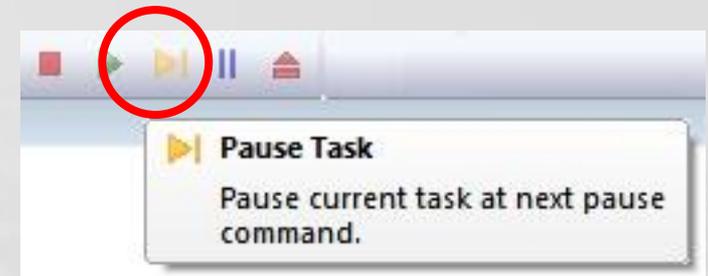
`PauseTask <task>`

- **Pause Task** command works only if there is **Pause** command

- Useful as a debugging tool

- ControlStudio

- Use the **Pause Task** button



# Continuing a Task

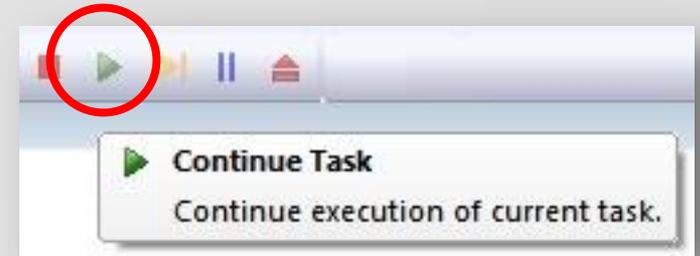
- **ContinueTask** resumes execution of an idled task

- Syntax/Example

`ContinueTask MyTask.prg`

- ControlStudio

- Use the **Continue Task (Run)** button



# Aborting a Running Task

- **KillTask** command aborts execution of a running task
- Syntax/Example  
`KillTask MyTask.prg`
- All attached motion elements are stopped and detached
- All events are cancelled
- Files that were opened by an aborted task will not be closed by the KillTask command.



# Running a Task Repeatedly (Loops)

- A program can be repeated a specific amount of times

- Syntax

```
StartTask <task> {NumberOfLoops = <number>}
```

- -1 = Unlimited number of loops
- 1 to 32768 = number of times the task is executed
- If *number* is not entered, it defaults to 1

- Example

```
StartTask MyTask.prg NumberOfLoops = 3
```

# Multitasking

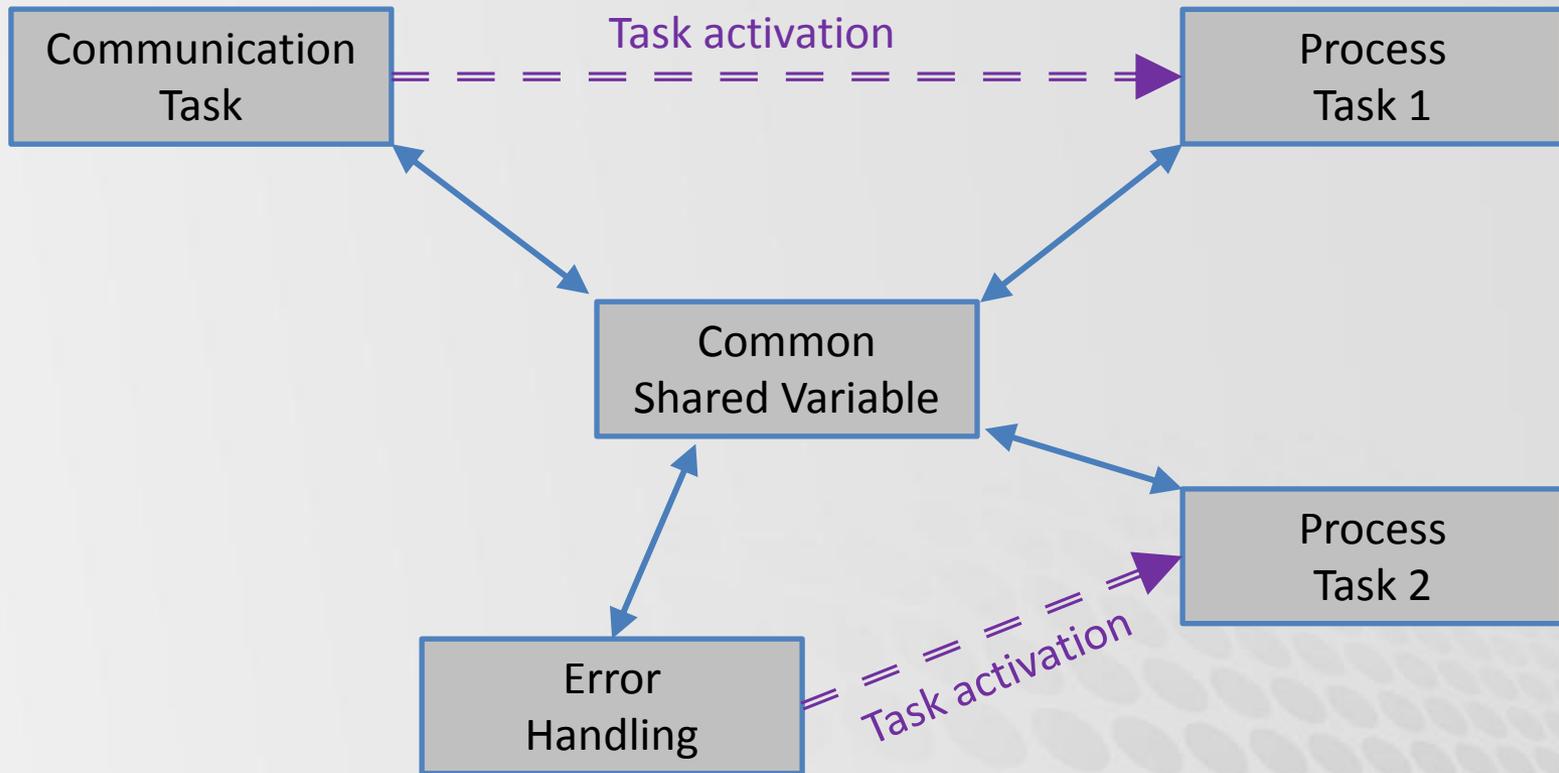
# Multitasking

- softMC can run up to 256 programs at one time
- Use multiple tasks when application has multiple processes that are essentially independent of each other
- Separate concurrent tasks should be used:
  - To simplify a system
  - To perform an operation that is completely independent from the main task
  - To run an independent processes on the machine
  - To run an operation at a higher priority level than the main task
  - To handle errors
  - To perform communication task
    - Example: teach pendant sends/receives commands while another executes

# Multitasking

- Multiple tasks can run independently
- If a machine is simple to control, keep the entire program in one task
- Do not split control of an axis or group across tasks
  
- Use multitasking when different parts/processes of a machine operate mostly independently of each other. Some control between tasks may be required, such as one task starting or stopping another
  
- Use the main task for machine initialization and controlling the other tasks, and use other tasks for programming normal machine operation
  - For example, use **Main.prg** to initialize the machine and to start **Pump.prg**, **Conveyor.prg**, and **Operator.prg**.
  
- Use different tasks to control different operational modes: one for power up, one for setup, one for normal operation, and another for when problems occur

# Task Interactions



# Semaphores

- Used for synchronization and mutual exclusion
- Semaphores are global, defined with **Common Shared**
- A semaphore is given/released by **SemaphoreGive**
- A semaphore is taken/consumed by **SemaphoreTake**

# Priority

# Multitasking and Task Priority

- The **operating system** (Linux RT) provides system resources based on two criteria:
  - Task priority level
  - Time slice
- Highest priority task always runs first
- Time slice (round-robin scheduling)
  - Operating system divides resources equally when multiple tasks have same priority level
  - The time slice is one millisecond in duration
  - A low priority task cannot interrupt a high priority task

# Assigning Task Priority Level

- The priority level of the task is assigned when it is loaded using **StartTask**

```
StartTask <task> {Priority = <priority>}
```

- Example:

```
StartTask MyTask.prg Priority = 4
```

- Level 1 – Highest priority level - Use with caution!
- Level 2 is default for Terminal – softMC command line
- Level 3 is default for Events
- Level 16 – Lowest priority level - Default for Tasks

# Multitasking – Relinquishing Computing Resources

- Tasks relinquish CPU resources when...
  - the task is terminated
  - the task is suspended
  - the task is idled
- Terminating tasks
  - When the task is completed
  - When any task uses the KillTask command
- Idled tasks
  - When any task uses the IdleTask
- Suspended tasks
  - **Note:** a task that is suspended is still running
  - When the task waits for a resource (e.g., motion element, semaphore)
  - Sleep command suspends task

# Task Status

# Task States

- **Ready**
  - Task is loaded and waiting for StartTask command
- **Running**
  - Task has been started by a StartTask command
- **Suspended**
  - Task is waiting for a resource.
  - Task is waiting for a motion to be completed
- **Idled**
  - A loaded task has executed the current command and does not continue until a ContinueTask command is issued
- **Terminated**
  - Task has finished running
  - Task has been aborted by KillTask command

# Query Task Status

- Query the state and priority of all tasks loaded in the system

- From the terminal

`?TaskList`

- Query the current status of a task that is loaded in memory

- From the terminal

`Task.Status`

`State <state>: <description> Error <last error number>`

`Source <line of source code>`

`Task.State`

Returns the numeric value of the task state

# Task States Numeric Values

- Returned *<state>* values:
  - 1 = Running
  - 2 = Stopped, due to IdleTask
  - 4 = Stopped, due to run-time error
  - 5 = Terminated
  - 7 = Ready, after Load
  - 10 = Killed, after KillTask or End Program

# Debugging

# Debugging

- Task must be in one of the following states:
  - Ready (7)
  - Stopped (2)
  - Error (4)
  - Killed (10)

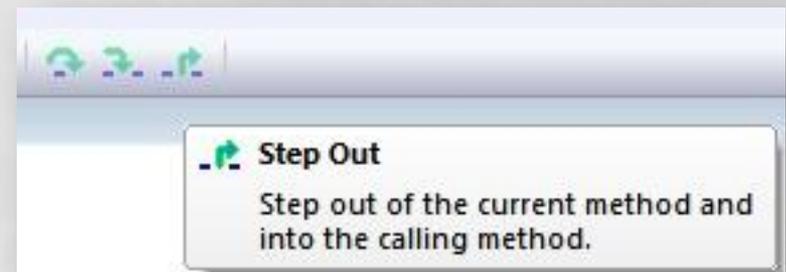
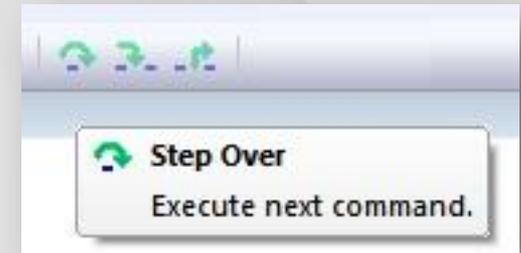
# Breakpoints

- Inserts a breakpoint in the task or library functions/subroutines at the specified program line number
- When a breakpoint is reached, the task switches to the idle state
- Program execution is resumed by issuing the ContinueTask command, or by using the program debugging commands: StepIn, StepOver, and StepOut



# Step Commands

- Step Over
  - Step Into
  - Step out
  - Break point
- 
- A step command executes one line of code
  - Depending on command, it will skip over the subroutine, step into the subroutine, or exit the subroutine



# Finding Source of Error in a Stopped Program

- **BACKTRACE** retrieves the function calls (source lines) that produced the error

The screenshot displays a development environment with three main windows:

- Message Log:** Shows an error message: "Error: 20020, 'MASTER COULD NOT FIND ANY SLAVES', Task: ETHERCAT.LIB, EC\_SETUP.PRG, Line: 2075, Module: User Exception".
- Terminal:** Shows the execution of the command `-->backtrace ec_setup.prg` and its output:

```
Stack EC_SETUP.PRG State 4
Source line : 2075 File : ETHERCAT.LIB Sub name : EC_ETHERCAT_INIT
Source line : 44 File : EC SETUP.PRG Sub name : PROGRAM
```
- Source Code:** Two windows show the source code. The left window shows the code around line 2075, where an exception is thrown: `throw EC_MASTER_NO_SLAVES_FOUND`. The right window shows the code around line 44, where the EtherCAT library is initialized: `numOfSlaves = EC_ETHERCAT_INIT`.

Red arrows point from the backtrace output in the terminal to the corresponding lines in the source code windows. A blue dashed arrow points from the error message in the message log to the backtrace command in the terminal.

# Error History

- `?errorhistory` returns the error log file, which contains the last 512 errors that have occurred in the system. Errors are saved in the Flash disk.

```
Terminal (Working folder: "C:\BMSD projects\tmp")
-->
-->
-->?errorhistory
```

Date	Time	Severity	Code	Task	File	Line/Addr.	Module	Message
10/12/2014	18:55:46.489	Note	4014	none			File_System	"Time-out during file transfer"
15/12/2014	11:46:33.309	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	11:46:33.341	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	11:46:33.377	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	11:49:21.933	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	11:49:21.969	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	11:49:22.005	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	11:57:48.563	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	11:57:48.599	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	11:57:48.635	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	12:07:34.746	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:07:34.778	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	12:07:34.814	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	12:16:36.161	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:33:41.120	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:33:41.156	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	12:33:41.192	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	12:34:11.433	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:36:32.917	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	12:36:35.270	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:38:36.977	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	12:42:00.398	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	12:54:47.183	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:54:47.215	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	12:54:48.264	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	12:55:28.212	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	12:55:40.856	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	12:56:10.311	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:58:35.238	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:59:04.693	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	12:59:05.741	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	12:59:05.777	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	13:00:59.669	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	13:28:58.476	Note	19006	none			Motion bus	"Drive 1 reports warning"
15/12/2014	13:28:58.512	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	13:28:58.548	Note	19006	none			Motion bus	"Drive 3 reports warning"
15/12/2014	13:30:41.111	Note	19006	none			Motion bus	"Drive 2 reports warning"
15/12/2014	13:51:22.448	Note	19006	none			Motion bus	"Drive 3 reports warning"

**END**