1.11 Interrupts

Introduction

Interrupts are program-defined events, identified by *interrupt numbers*. An interrupt occurs when an *interrupt condition* is true. Unlike errors, the occurrence of an interrupt is not directly related to (synchronous with) a specific code position. The occurrence of an interrupt causes suspension of the normal program execution and control is passed to a *trap routine*.

Even though the robot immediately recognizes the occurrence of an interrupt (only delayed by the speed of the hardware), its response – calling the corresponding trap routine – can only take place at specific program positions, namely:

- when the next instruction is entered,
- any time during the execution of a waiting instruction, for example `WaitUntil`,
- any time during the execution of a movement instruction, for example `MoveL`.

This normally results in a delay of 2-30 ms between interrupt recognition and response, depending on what type of movement is being performed at the time of the interrupt.

The raising of interrupts may be *disabled* and *enabled*. If interrupts are disabled, any interrupt that occurs is queued and not raised until interrupts are enabled again. Note that the interrupt queue may contain more than one waiting interrupt. Queued interrupts are raised in *FIFO* order (first in, first out). Interrupts are always disabled during the execution of a trap routine.

When running stepwise and when the program has been stopped, no interrupts will be handled. Interrupts in queue at stop will be thrown away and any interrupts generated under stop will not be dealt, except for safe interrupts, see *Safe Interrupt on page 71*.

The maximum number of defined interrupts at any one time is limited to 100 **per program task**.

Programming principles

Each interrupt is assigned an interrupt identity. It obtains its identity by creating a variable (of data type intnum) and connecting this to a trap routine.

The interrupt identity (variable) is then used to order an interrupt, that is to specify the reason for the interrupt. This may be one of the following events:

- An input or output is set to one or to zero.
- A given amount of time elapses after an interrupt is ordered.
- A specific position is reached.

When an interrupt is ordered, it is also automatically enabled, but can be temporarily disabled. This can take place in two ways:

- All interrupts can be disabled. Any interrupts occurring during this time are placed in a queue and then automatically generated when interrupts are enabled again.

Continues on next page

- Individual interrupts can be deactivated. Any interrupts occurring during this time are disregarded.

Connecting interrupts to trap routines

Instruction	Used to
CONNECT	Connect a variable (interrupt identity) to a trap routine

Ordering interrupts

Instruction	Used to order
ISignalDI	An interrupt from a digital input signal
ISignalDO	An interrupt from a digital output signal
ISignalGI	An interrupt from a group of digital input signals
ISignalGO	An interrupt from a group of digital output signals
ISignalAI	An interrupt from an analog input signal
ISignalAO	An interrupt from an analog output signal
ITimer	A timed interrupt
TriggInt	A position-fixed interrupt (from the Motion pick list)
IPers	An interrupt when changing a persistent.
IError	Order and enable an interrupt when an error occurs
IRMQMessage [i]	An interrupt when a specified data type is received by a RAPID Message Queue

[i] Only if the robot is equipped with the option *FlexPendant Interface*, *PC Interface*, or *Multitasking*.

Cancelling interrupts

Instruction	Used to
IDelete	Cancel (delete) an interrupt

Enabling/disabling interrupts

Instruction	Used to
ISleep	Deactivate an individual interrupt
IWatch	Activate an individual interrupt
IDisable	Disable all interrupts
IEnable	Enable all interrupts

Interrupt data

Instruction	Used to
GetTrapData	in a trap routine to obtain all information about the interrupt that caused the trap routine to be executed.
ReadErrData	in a trap routine, to obtain numeric information (domain, type and number) about an error, a state change, or a warning, that caused the trap routine to be executed.

Continues on next page

Data type of interrupts

Data type	Used to
`intnum`	Define the identity of an interrupt.
`trapdata`	Contain the interrupt data that caused the current TRAP routine to be executed.
`errtype`	Specify an error type (gravity)
`errdomain`	Order and enable an interrupt when an error occurs.
`errdomain`	Specify an error domain.

Safe Interrupt

Some instructions, for example `ITimer` and `ISignalDI`, can be used together with Safe Interrupt. Safe Interrupts are interrupts that will be queued if they arrive during stop or stepwise execution.The queued interrupts will be dealt with as soon as continuos execution is started, they will be handled in *FIFO* order. Interrupts in queue at stop will also be dealt with. The instruction `ISleep` can not be used together with safe interrupts.

Interrupt manipulation

Defining an interrupt makes it known to the system. The definition specifies the interrupt condition and activates and enables the interrupt.

Example:

```
VAR intnum sig1int;
ISignalDI di1, high, sig1int;
```

An activated interrupt may in turn be deactivated (and vice versa).

During the deactivation time, any generated interrupts of the specified type are thrown away without any trap execution.

Example:

```
! deactivate
ISleep sig1int;

! activate
IWatch sig1int;
```

An enabled interrupt may in turn be disabled (and vice versa).

During the disable time, any generated interrupts of the specified type are queued and raised first when the interrupts are enabled again.

Example:

```
! disable
IDisable sig1int;

1 enable
IEnable sig1int;
```

Deleting an interrupt removes its definition. It is not necessary to explicitly remove an interrupt definition, but a new interrupt cannot be defined to an interrupt variable until the previous definition has been deleted.

Continues on next page

Example:

```
IDelete sig1int;
```

Trap routines

Trap routines provide a means of dealing with interrupts. A trap routine can be connected to a particular interrupt using the CONNECT instruction. When an interrupt occurs, control is immediately transferred to the associated trap routine (if any). If an interrupt occurs, that does not have any connected trap routine, this is treated as a fatal error, that is causes immediate termination of program execution.

Example:

```
VAR intnum empty;
VAR intnum full;
PROC main()
  ! Connect trap routines
  CONNECT empty WITH etrap;
  CONNECT full WITH ftrap;
  ! Define feeder interrupts
  ISignalDI di1, high, empty;
  ISignalDI di3, high, full;
  ...
  ! Delete interrupts
  IDelete empty;
  IDelete full;
ENDPROC
! Responds to “feeder empty” interrupt
TRAP etrap
  open_valve;
  RETURN;
ENDTRAP
! Responds to “feeder full” interrupt
TRAP ftrap
  close_valve;
  RETURN;
ENDTRAP
```

Several interrupts may be connected to the same trap routine. The system variable INTNO contains the interrupt number and can be used by a trap routine to identify an interrupt. After the necessary action has been taken, a trap routine can be terminated using the RETURN instruction or when the end (ENDTRAP or ERROR) of the trap routine is reached. Execution continues from the place where the interrupt occurred.