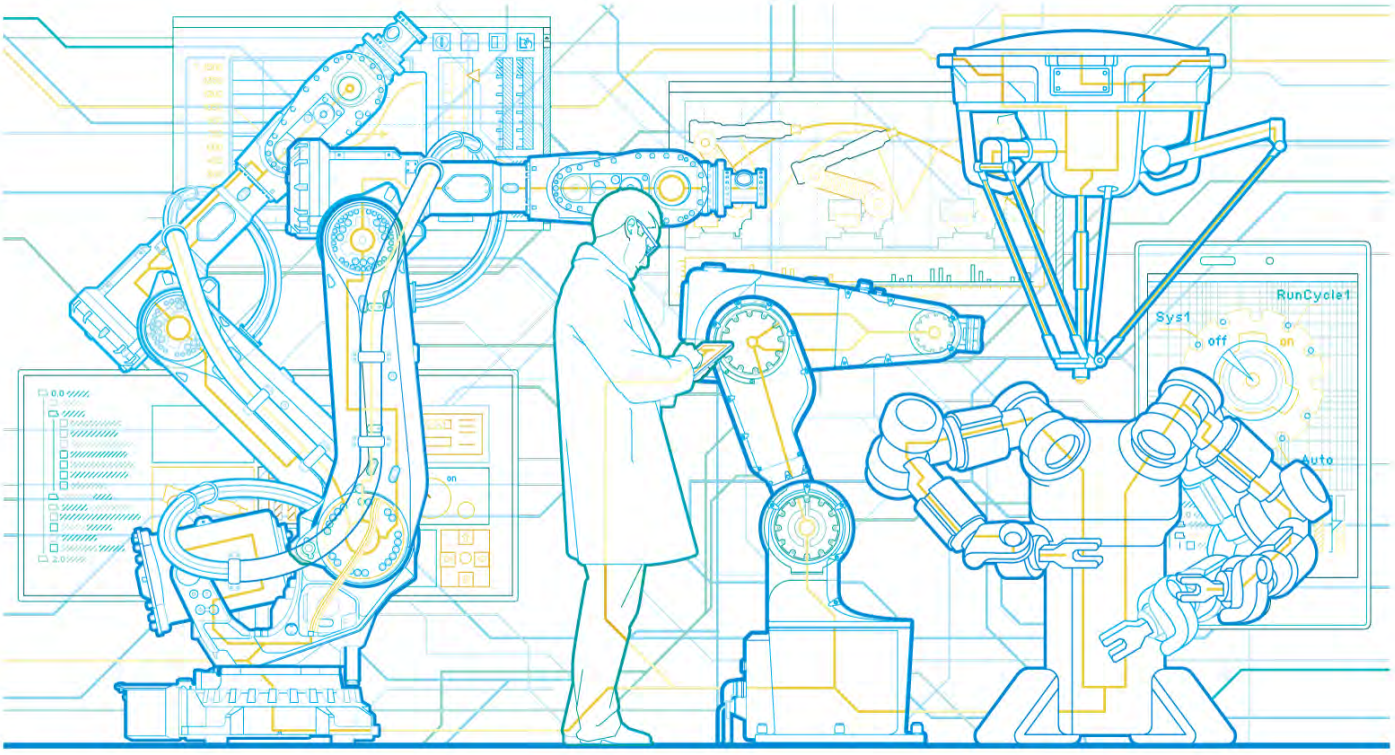


Application manual
Continuous Application Platform

Trace back information:
Workspace R17-1 version a6
Checked in 2017-03-21
Skribenta version 5.1.011

Application manual

Continuous Application Platform

RobotWare 6.05

Document ID: 3HAC050990-001

Revision: C

ABB AB, Robotics
Robotics and Motion
Se-721 68 Västerås
Sweden

Table of contents

This page is intentionally left blank

Overview of this manual

About this manual

This manual describes the option *Continuous Application Platform* and contains instructions for the configuration.

Who should read this manual?

This manual is intended for:

- Personnel responsible for installations and configurations of fieldbus hardware/software
- Personnel responsible for I/O system configuration
- System integrators

Prerequisites

The reader should have the required knowledge of:

- Mechanical installation work
- Electrical installation work
- System parameter configuration

Organization of chapters

The manual is organized in the following chapters:

Chapter	Contents
1	This chapter consists of a number of RAPID instructions and data types that make development of continuous applications easier, faster, and more robust.
2	This chapter consists of instructions and functionality of CAP.
3	This chapter lists the programming examples of CAP.
4	This chapter consists of different instructions and data types of CAP.

References

References	Document ID
Application manual - Arc and Arc Sensor	*3HAC050988-001*
Application manual - Controller software IRC5	*3HAC050798-001*
Technical reference manual - RAPID Instructions, Functions and Data types	*3HAC050917-001*
Technical reference manual - RAPID overview	*3HAC050947-001*

Revisions

Revision	Description
-	Released with RobotWare 6.0.
A	Released with RobotWare 6.01. Minor corrections.

Continues on next page

Continued

Revision	Description
B	Released with RobotWare 6.04. • Minor corrections. • The RAPID instructions, functions, and data types are moved to *Technical reference manual - RAPID Instructions, Functions and Data types*.
C	Released with RobotWare 6.05. • Minor corrections.

Product documentation, IRC5

Categories for user documentation from ABB Robotics

The user documentation from ABB Robotics is divided into a number of categories. This listing is based on the type of information in the documents, regardless of whether the products are standard or optional.

All documents listed can be ordered from ABB on a DVD. The documents listed are valid for IRC5 robot systems.

Product manuals

Manipulators, controllers, DressPack/SpotPack, and most other hardware is delivered with a **Product manual** that generally contains:

- Safety information.
- Installation and commissioning (descriptions of mechanical installation or electrical connections).
- Maintenance (descriptions of all required preventive maintenance procedures including intervals and expected life time of parts).
- Repair (descriptions of all recommended repair procedures including spare parts).
- Calibration.
- Decommissioning.
- Reference information (safety standards, unit conversions, screw joints, lists of tools).
- Spare parts list with exploded views (or references to separate spare parts lists).
- Circuit diagrams (or references to circuit diagrams).

Technical reference manuals

The technical reference manuals describe reference information for robotics products.

- *Technical reference manual - Lubrication in gearboxes*: Description of types and volumes of lubrication for the manipulator gearboxes.
- *Technical reference manual - RAPID overview*: An overview of the RAPID programming language.
- *Technical reference manual - RAPID Instructions, Functions and Data types*: Description and syntax for all RAPID instructions, functions, and data types.
- *Technical reference manual - RAPID kernel*: A formal description of the RAPID programming language.
- *Technical reference manual - System parameters*: Description of system parameters and configuration workflows.

Continues on next page

Application manuals

Specific applications (for example software or hardware options) are described in **Application manuals**. An application manual can describe one or several applications.

An application manual generally contains information about:

- The purpose of the application (what it does and when it is useful).
- What is included (for example cables, I/O boards, RAPID instructions, system parameters, DVD with PC software).
- How to install included or required hardware.
- How to use the application.
- Examples of how to use the application.

Operating manuals

The operating manuals describe hands-on handling of the products. The manuals are aimed at those having first-hand operational contact with the product, that is production cell operators, programmers, and trouble shooters.

The group of manuals includes (among others):

- *Operating manual - Emergency safety information*
- *Operating manual - General safety information*
- *Operating manual - Getting started, IRC5 and RobotStudio*
- *Operating manual - IRC5 Integrator's guide*
- *Operating manual - IRC5 with FlexPendant*
- *Operating manual - RobotStudio*
- *Operating manual - Trouble shooting IRC5*

Safety

Safety of personnel

When working inside the robot controller it is necessary to be aware of voltage-related risks.

A danger of high voltage is associated with the following parts:

- Devices inside the controller, for example I/O devices, can be supplied with power from an external source.
- The mains supply/mains switch.
- The power unit.
- The power supply unit for the computer system (230 VAC).
- The rectifier unit (400-480 VAC and 700 VDC). Capacitors!
- The drive unit (700 VDC).
- The service outlets (115/230 VAC).
- The power supply unit for tools, or special power supply units for the machining process.
- The external voltage connected to the controller remains live even when the robot is disconnected from the mains.
- Additional connections.

Therefore, it is important that all safety regulations are followed when doing mechanical and electrical installation work.

Safety regulations

Before beginning mechanical and/or electrical installations, ensure you are familiar with the safety regulations described in *Operating manual - General safety information*[1].

[1] This manual contains all safety instructions from the product manuals for the manipulators and the controllers.

This page is intentionally left blank

1 Continuous Application Platform

Introduction

The Continuous Application Platform (CAP) consists of a number of RAPID instructions and data types that make development of continuous applications easier, faster, and more robust.

The basic idea of CAP is to separate motion synchronization and application control by moving the application control from the controller software kernel to an application layer in RAPID. By this two things are achieved:

- The CAP core is robust and generic.
- The application layer is easier to customize.

CAP is a process event dispatcher. The core of CAP is a generic state engine from which the application builder (developer) can choose to subscribe specific process events (`ICap`).

This page is intentionally left blank

2 Functionality of CAP

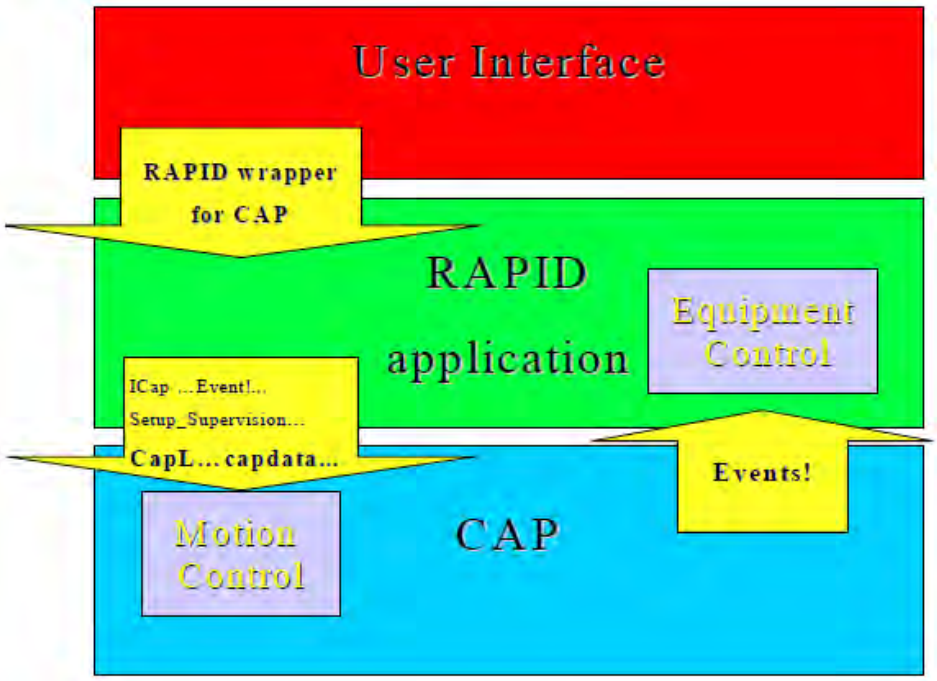
Description of CAP

With CAP it is possible to control a continuous application process and at the same time synchronize that process with the TCP movement of a robot.

The synchronization between motion and application layer is handled via events from the CAP core triggering trap routines in RAPID (*Predefined events on page 27*).

CAP enables the RAPID user to order supervision of I/O signals depending on the motion of the robot (*Supervision on page 18*).

For synchronization of motion and process, the process is divided into different phases. These phases tell the CAP core which I/O signals to supervise and in which way (*Process phases on page 16*).



xx1200000163

Continues on next page

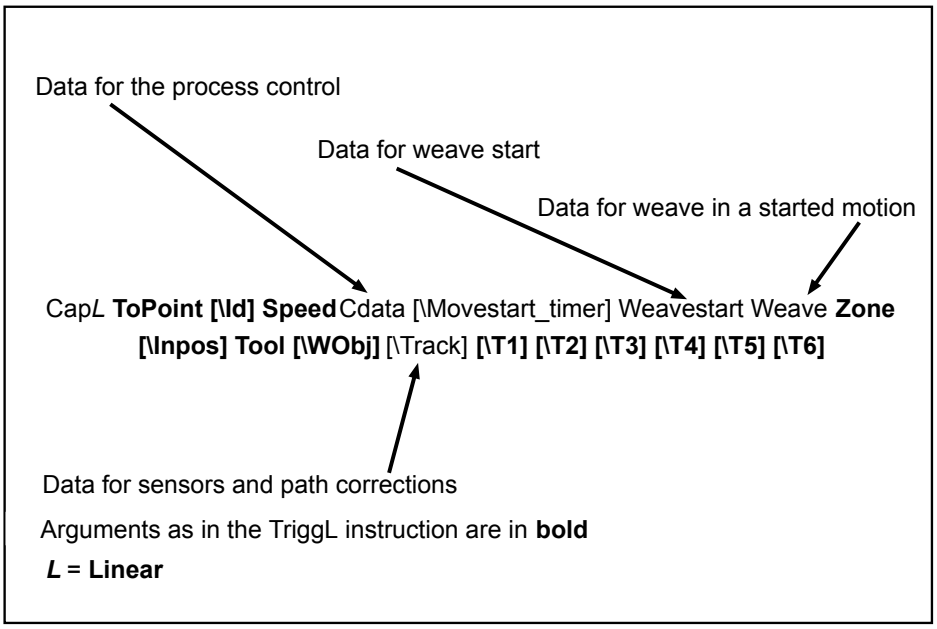
2.1 Robot movement

Instructions and motion

A CAP motion instruction (`CapL` or `CapC`) is similar to other motion instructions (for example, `MoveL`, `TriggL`). Compared to the `TriggL` instruction it contains also the information necessary for CAP. That information is given through the arguments `Cdata`, `Weavestart`, `Weave` and optional parameter `\Track`.

The motion control is handled by a state engine based on the general process graph class.

The CAP process is path persistent, which means that it is active over several CAP motion instructions from the first CAP instruction to the next CAP instruction containing the `Cdata.last_instr` set to `TRUE` (see `capdata` in *Technical reference manual - RAPID Instructions, Functions and Data types*).



xx1200000164

The `Speed` argument in the instruction is only valid during step-wise execution (forward or backward) and flying end from the point where the application process has been stopped. The CAP process will in this case automatically be inhibited. During normal execution, the process speed in different phases of the process is defined in the component `Cdata` of the type `capdata`.

For more information on programming CAP motion instructions see *Programming examples on page 41*.

Process phases

The CAP process is divided into four different phases to synchronize the robot movement with the application process:

- Pre_start
- Main

Continues on next page

- Post1
- Post2

The process phases are tightly bound to the supervision phases (*Supervision on page 18*).

During the process phases several events are generated by CAP, which can be connected to TRAP routines in the RAPID layer.

2.2 Supervision

Introduction to supervision

Supervision is used to control signals during the process and perform proper actions if some supervised signal fails.

Supervision is ordered at RAPID level (see `SetupSuperv` in *Technical reference manual - RAPID Instructions, Functions and Data types*).

Supervision phases

As mentioned in *Process phases on page 16*, the CAP process is divided in four phases. Those phases are tightly bound to the supervision phases. Supervision is divided in twelve supervision phases:

- PRE
- PRE_START (corresponds to CAP process phase Pre)
- END_PRE
- START
- MAIN (corresponds to CAP process phase Main)
- END MAIN
- START_POST1
- POST1 (corresponds to CAP process phase Post1)
- END_POST1
- START_POST2
- POST2 supervision (corresponds to CAP process phase Post2)
- END_POST2

The different supervision phases use corresponding supervision lists, which book-keep the signal supervisions ordered by the RAPID developer. The naming of the lists is similar to that of the corresponding phases, for example, the list for the PRE supervision phase is called SUPERV_PRE (see `SetupSuperv` in *Technical reference manual - RAPID Instructions, Functions and Data types*). The maximum number of signals that can be supervised during each phase is 32.

There are two different types of supervision phases:

- Handshake supervision.
- Status supervision.

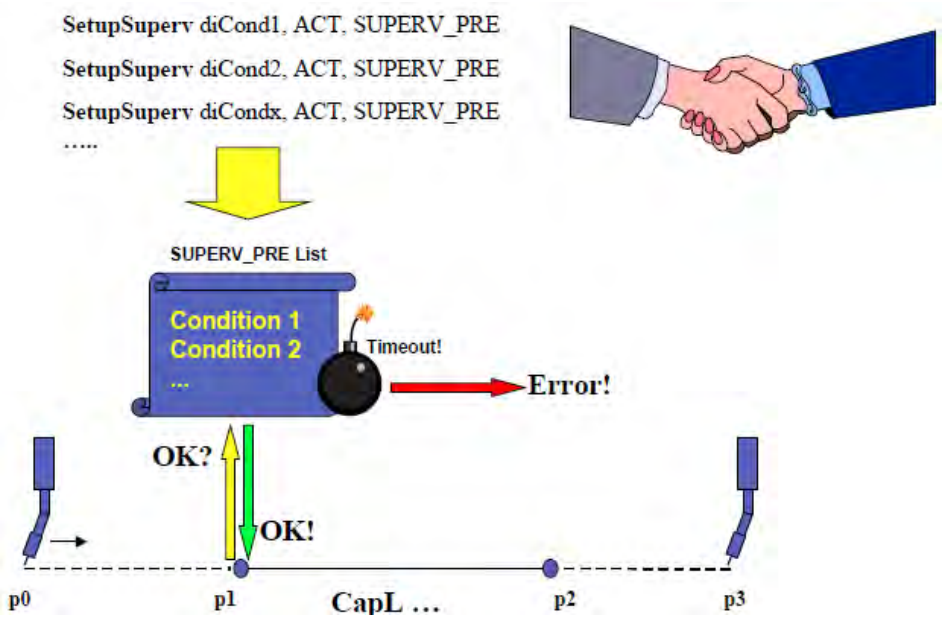
Handshake supervision

There is a handshake supervision phase before and after each status supervision phase. They prevent the process phase from starting until all conditions in the handshake supervision list are fulfilled.

It is possible to use a time-out for handshake supervision. If a time-out is used and not all supervision requests are fulfilled when it expires, an ERROR is generated. The time-out can be set to last forever, that is, the CAP process will be waiting for all supervision request to be fulfilled. The time-outs are specified in

Continues on next page

`supervtimeouts` which is part of the `capdata` (see `supervtimeouts` in *Technical reference manual - RAPID Instructions, Functions and Data types*).



xx1200000165

The handshake supervision works as a threshold, to prevent the process from proceeding to the next phase. If no handshake supervision is set up that phase is skipped.

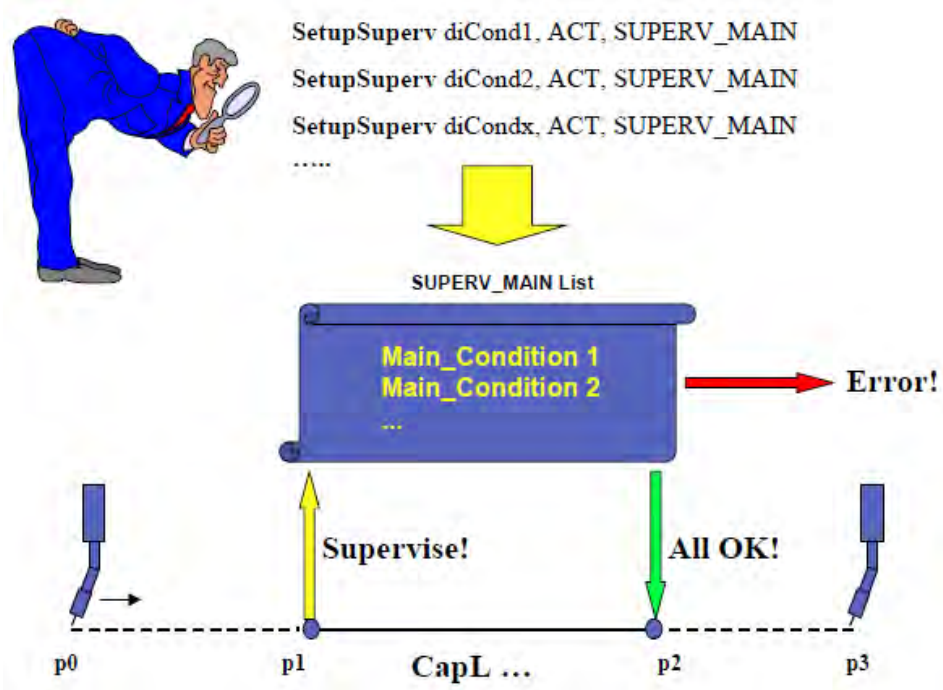
These are the handshake supervision phases.

- PRE
- END_PRE
- START
- END MAIN
- START_POST1
- END_POST1
- START_POST2
- END_POST2

Continues on next page

Status supervision

During the status supervision phases all signals the user requested supervision on (`SetupSuperv`) are supervised (see figure below).



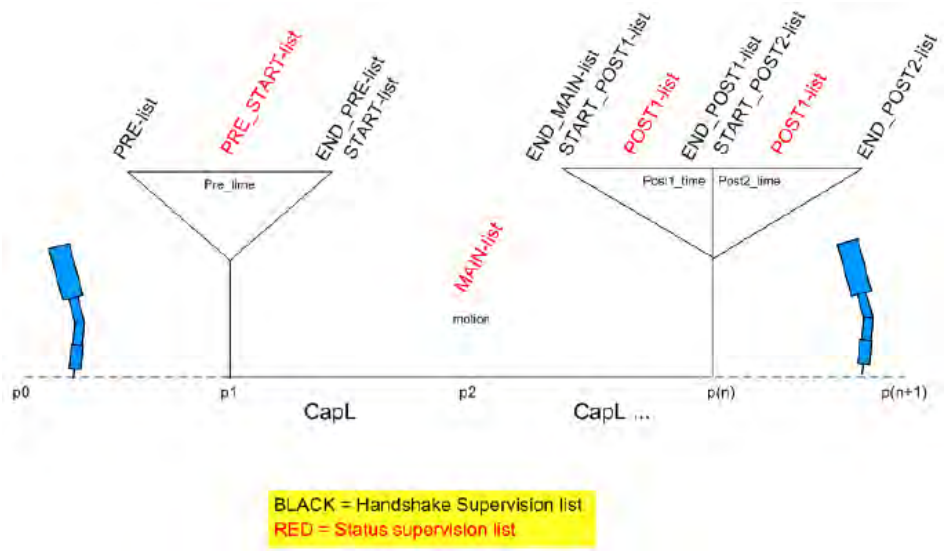
xx1200000166

The component `proc_times` in `capdata` defines the duration of the phases PRE_START, POST1, and POST2 will be. If supervision is requested during any of these phases, the duration time for each phase must be bigger than zero; otherwise the supervision will fail. No time has to be specified for the MAIN phase, because this time is defined by the movement of the robot.

These are the status supervision phases.

- PRE_START
- MAIN
- POST1
- POST2

Continues on next page



xx1200000167

2.3 Supervision and process phases

Phases

The PRE_START phases, the POST1 phases and the POST2 phases are common to one single CAP process path, that is:

- the first CAP instruction that starts or restarts the CAP process is the only one that has PRE_START supervision phases. At restart the presence of these phases depends on the setting of pre_phase in the data type `restartblkdata`. See `restartblkdata` in *Technical reference manual - RAPID Instructions, Functions and Data types*.
- the last CAP instruction (`last_instr` set to `TRUE` in `capdata`) that terminates the CAP process is the only one that has POST1 phases and POST2 phases. See `capdata` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

PRE_START phases

When the robot reaches the start point of the path, all conditions in the PRE supervision list must be fulfilled for the process to be allowed to enter the PRE_START status supervision phase. If a time-out is specified and the conditions cannot be met within that time, the process is stopped, and an error is sent.

During the PRE_START phase all conditions defined in the PRE_START supervision list must be fulfilled. If some of these conditions fail, the process is stopped and an error message is sent.

After the PRE_START phase all conditions in the END_PRE supervision list must be fulfilled for the process to be allowed to enter the START handshake supervision phase. If a time-out is specified and the conditions cannot be met within that time, the process is stopped, and an error is sent.

When using *flying start* this phase will not be available, but the duration time can be used to create an ignition delay.

Summary

- Starts when all conditions in the PRE supervision list are met.
- Supervised by the PRE_START supervision list.
- Ends when all conditions in the END_PRE supervision list are met.

MAIN phases

All conditions in the START supervision list must be fulfilled for the process to be allowed to enter the MAIN status supervision phase. If a time-out is specified and the conditions cannot be met within that time, the process is stopped, and an error is sent.

During the MAIN phase all conditions defined in the MAIN supervision list must be fulfilled. If some of these conditions fail, the process is stopped and an error message is sent.

Continues on next page

All conditions in the END_MAIN supervision list must be fulfilled for the process to end the MAIN phases. If the conditions cannot be met within that time, the process is stopped, and an error is sent.

Summary

- Starts when all conditions in the START supervision list are met.
- Supervised by the MAIN supervision list.
- Ends when all conditions in the END_MAIN supervision list are met.

POST1 phase

All conditions in the START_POST1 supervision list must be fulfilled for the process to be allowed to enter the POST1 status supervision phase. If a time-out is specified for START_POST1 and the conditions cannot be met within that time, the process is stopped, and an error is sent.

During the POST1 phase all conditions defined in the POST1 supervision list must be fulfilled. If some of these conditions fail, the process is stopped and an error message is sent.

All conditions in the END_POST1 supervision list must be fulfilled for the process to end the POST1 phases. If the conditions cannot be met within that time, the process is stopped, and an error is sent.

This phase is not available for *flying start*.

Summary

- Starts when all conditions in the START_POST1 supervision list are met.
- Supervised by the POST1 supervision list.
- Ends when all conditions in the END_POST1 supervision list are met.

POST2 phase

All conditions in the START_POST2 supervision list must be fulfilled for the process to be allowed to enter the POST2 status supervision phase. If a time-out is specified for START_POST2 and the conditions cannot be met within that time, the process is stopped, and an error is sent.

During the POST2 phase all conditions defined in the POST2 supervision list must be fulfilled. If some of these conditions fail, the process is stopped and an error message is sent.

This phase is not available for *flying start*.

Summary

- Starts when all conditions in the START_POST2 supervision list are met.
- Supervised by the POST2 supervision list.
- Ends when all conditions in the END_POST2 supervision list are met.

2.4 Motion delay

Description

Motion delay gives the user the possibility to delay the start of the motion. This is useful for applications, for example laser cutting, where the movement cannot be started before the material has been burned through. The time for the motion delay is specified in `capspeeddata`. See `capspeeddata` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

This functionality is not available for *flying start*.

2.5 Programming recommendations

Corner zones

A sequence of CAP instructions shall have corner zones (for example, z10) on the path.

For example:

```
MoveL p10,v100,fine,tool;
CapL p20,v50,cdata,nowvst,nowv,z20,tool;
CapC p30,p40,v50,cdata,nowvst,nowv,z20,tool;
CapL p50,v50,cdata,nowvst,nowv,z20,tool;
CapL p60,v50,cdata,nowvst,nowv,fine,tool;
MoveL p70,v100,fine,tool;
```

If the last movement instruction before the first CAP instruction has a zone, CAP will start the application process with a *flying start*.

If the last instruction of a sequence of CAP instructions has a zone, CAP will end the application process with a *flying end*.

Within a sequence of CAP instructions, avoid logical instructions that take long time. That will cause the errors *50024 Corner path failure* and *110013 Application process interrupted*, which means, that a corner zone is converted to a stop point, and the application process is interrupted and restarted with the next CAP instruction.

2.6 Program execution

Corner zones

If `last_instr` is set to `TRUE` in `capdata` in the middle of a sequence of CAP instructions, the application process is finished with all phases as described in *Process phases on page 16*. Which phases are executed depends on the presence of *flying end*. The following CAP instruction will start the process again, with all phases as described in *Process phases on page 16*. Which phases are executed, depends on the presence of *flying start*.

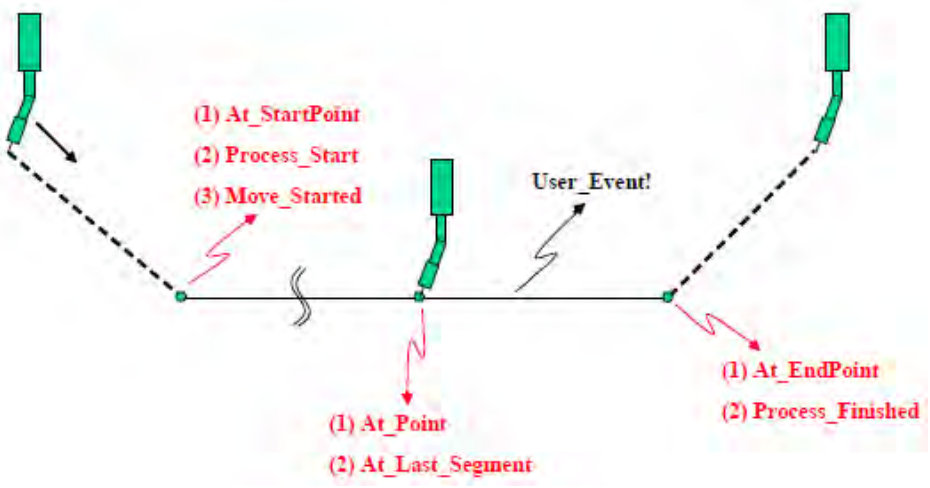
If a stop point occurs in the middle of a sequence of CAP instructions without `last_instr` set to `TRUE` in `capdata`, the application process will not be interrupted, the program execution will proceed to the next CAP instruction in advance (prefetch), and the movement will execute a stop point.

If execution of logical instructions in the middle of a sequence of CAP instructions take so long time, that a programmed corner path is converted to a stop point (*50024 Corner path failure*), the application process is interrupted (*110013 Application process interrupted*) without executing the phases described in *Process phases on page 16*.

2.7 Predefined events

Description

CAP can connect to RAPID interrupt routines with a number of predefined events, which occur during the CAP process. To do this, the RAPID instruction `ICap` is used before running the first CAP movement instruction. This enables the user to synchronize external equipment with the robot movement using RAPID. See `ICap` in *Technical reference manual - RAPID Instructions, Functions and Data types*.



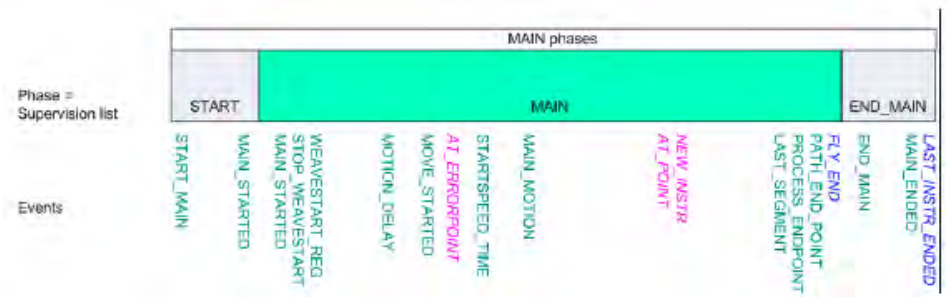
xx1200000168

2.8 Coupling between phases and events

PRE_START supervision phases

xx1200000169

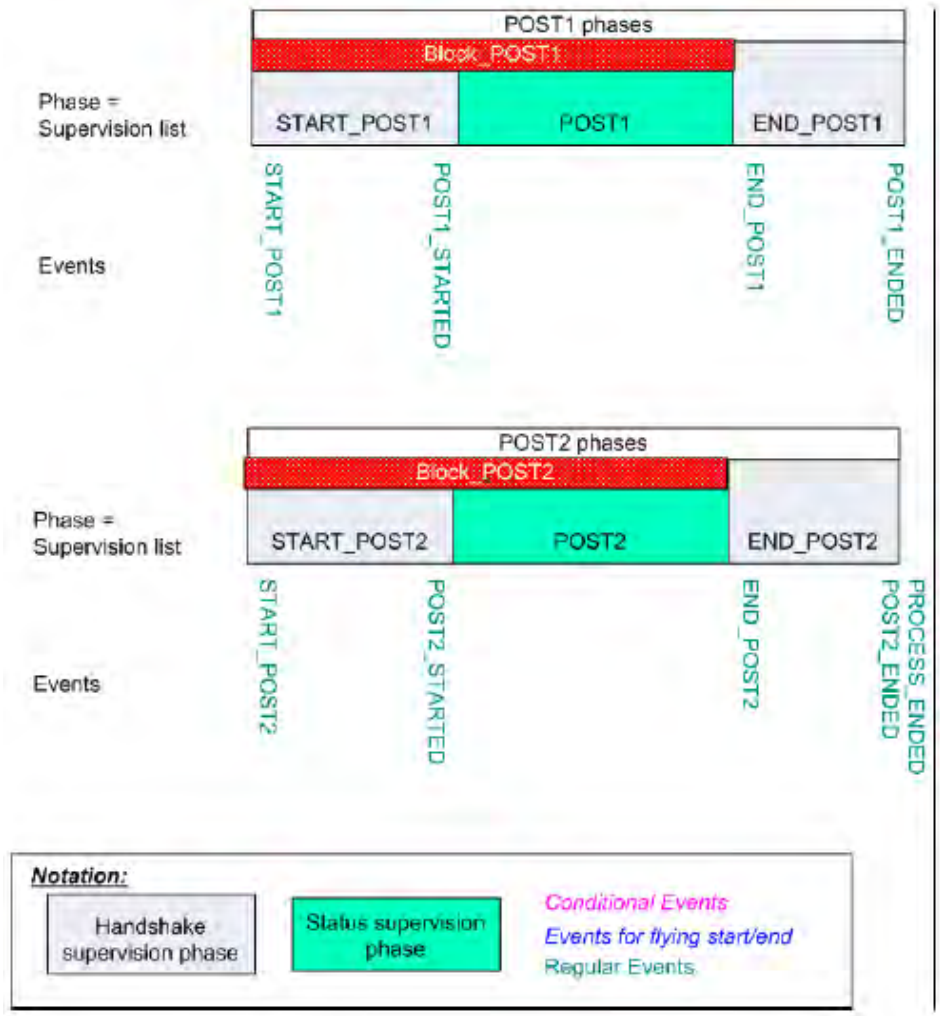
MAIN supervision phases



xx1200000170

Continues on next page

POST1 and POST2 supervision phases



xx1200000171

User events

The CAP movement instructions, `CapL` and `CapC`, offer the possibility to define up to six trigger events (switches `\T1` to `\T8`). These trigger events can be coupled to CAP movement instructions with `TriggIO`, `TriggEquip` or `TriggInt`. See `CapL` and `CapC` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

2.9 Error handling

Description

Two different types of error can occur during `CapL` or `CapC`:

- Recoverable errors: these errors can be handled in a RAPID error handler (see error handling for `CapL` in *Technical reference manual - RAPID Instructions, Functions and Data types*). The system variable ERRNO is set and the user can check ERRNO to get information about which error occurred and choose the adequate recovery measures. For recoverable errors it is possible to use the RAPID instruction `RETRY` in the error handler. An error message is generated.
- Fatal errors: if such an error occurs, the robot controller has to be restarted. A fatal error message is generated.

Continues on next page

2.9.1 Recoverable errors

Introduction

Recoverable errors can be handled in a RAPID error handler. The CAP user can then choose to RETRY a several number of times depending on the application and the error. If for example, the arc in an arc welding application does not strike the first time, it makes sense to retry arc ignition a several number of times. If these attempts are unsuccessful the error may be raised to the next level of RAPID (RAISE) or, if it is in a NOSTEPIN/NOVIEW module, to user level (`RaiseToUser`) - see examples below.

Errors from CapL and CapC

Errors from the instructions `CapL` and `CapC` are CAP specific (see `CapL` and `CapC` in *Technical reference manual - RAPID Instructions, Functions and Data types*). That means, that those error codes have to be translated to application specific error codes in the error handler, to make it easier for users of that application to understand the error message. After remapping of the error the new error code is raised to the user (RAISE new_err_code) - see *Example 1 on page 32*.

As a consequence these errors should be converted to application specific errors, depending on which application CAP is used for. To achieve this the `ERRNO` has to be checked in the error handler. See *Example 1 on page 32*.

If for example a supervised signal fails, for example, in the main supervision phase, the enduser should not get a general `CAP_MAIN_ERR` error. The application layer should return a more specific error, since this error depends on how CAP is used by the RAPID application. If several signals are supervised during a supervision phase, all these signals have to be checked in the application error handler to identify the error more specifically.

No error handler

If no error handler can be found or there is an error handler, but it does not handle the error - that is, none of the instructions `RETRY`, `TRYNEXT`, `RETURN` or `RAISE` are present in the error handler - the active motion path is cleared. That means, that neither *regain to path* nor *backing on the path* is possible. The robot movement starts from the current position of the TCP, which might result in a *path shortcut*.

Start phase supervision errors

If a START phase supervision error occurs during *flying start*, the movement is stopped at the end of the *START distance* and at restart the application process is handled like an ordinary restart after an error - with all user defined restart functionality like *scrape start*, *start delay*, etc.

Examples

Below are two examples of different error handling type. The type of error handling that is recommended is the one in example 2, where the CAP process survives and no extra code has to be executed in a retry from user level. See *Example 2 on page 33*.

Continues on next page

2.9.1 Recoverable errors
Continued

The SkipWarn instruction is used in the error handlers to prevent the CAP specific error from being sent to the error log. For an application (for example, Arc Welding) user CAP errors are not interesting. The errors shown in the event log shall be application specific.

Example 1

This is an example with RAPID modules, that are not NOSTEPIN/NOVIEW. If the error is RAISEd to the RAPID main routine the CAP process will exit.

A RETRY order in the error handler case MY_AW_ERR_1 will continue execution and make a retry on arcl_move_only. The value of example_count1 will be 2 when executing the CapL instruction after a retry from the higher level.

```
MODULE CAP_EXAMPLE1
VAR num example1_count:=0;
PROC main()
  MoveJ p10,v200,fine,tool0;
  arcl_move_only p11, v20, z20, tool0;
  ERROR
    TEST ERRNO
    CASE MY_AW_ERR_1:
      RETRY;
    CASE MY_AW_ERR_2:
      EXIT;
    DEFAULT:
      EXIT;
    ENDTEST
ENDPROC

LOCAL PROC arcl_move_only (robtarget ToPoint, speeddata Speed,
     zonedata Zone, PERS tooldata Tool \PERS wobjdata WObj \switch
     Corr)

example1_count := example1_count + 1;
CapL Topoint, Speed, IntCdata, IntWeavestart, IntWeave, Zone, Tool
     \wobj?wobj;
ERROR
  ResetIoSignals;

  IF no_of_retries > 0 THEN
    IF err_cnt < no_of_retries THEN
      err_cnt := err_cnt + 1;
      Skipwarn; !Remove CAP error from event log err_code :=
           new_aw_errMsg();
      RETRY; !*StartMoveRetry
    ELSE
      err_cnt := 0;
      Skipwarn;
      err_code := new_aw_errMsg();
      RAISE err_code;
      !Kills the CAP process, and raises mapped error
    ENDIF
```

Continues on next page

```
  ELSE
    Skipwarn;
    err_code := new_aw_errMsg();
    RAISE err_code;
      !Kills the CAP process, and raises mapped error
  ENDIF
ENDPROC

FUNC errnum new_aw_errMsg (\switch W)

  VAR errnum ret_code;

  TEST ERRNO

  CASE CAP_PRE_ERR:
    ! Check of signals here
    ret_code := AW_EQIP_ERR;
  ENDTEST

  RETURN ret_code;

ENDFUNC

ENDMODULE
```

Example 2

This is an example with one RAPID module, that is NOSTEPIN/NOVIEW, where main is located. The `arcl_move_only` procedure is located in a NOVIEW module. If the error is raised to main routine with the instruction `RaiseToUser \Continue` the CAP process is still active. The `RETRY` order in the error handler case MY_AW_ERR_1 will continue execution and make a retry directly on the `CapL` instruction a second time. The example_count1 will be 1 when executing the `CapL` instruction after a retry from the user level.

Note

The `RaiseToUser` instruction can only be used in a NOVIEW module.

```
MODULE CAP_EXAMPLE

VAR num example1_count:=0;

PROC main()
  MoveJ p10,v200,fine,tool0;
  arcl_move_only p11, v20, z20, tool0;

  ERROR
    TEST ERRNO
    CASE MY_AW_ERR_1:
      RETRY;
```

Continues on next page

2.9.1 Recoverable errors
Continued

```
    CASE MY_AW_ERR_2:
      EXIT;
    DEFAULT:
      EXIT;
    ENDTEST
ENDPROC
ENDMODULE

MODULE ARCX_MOVE_ONLY(NOSTEPIN, NOVIEW)
LOCAL PROC arcl_move_only(robtarget ToPoint, speeddata Speed,
     zonedata Zone, PERS tooldata Tool \PERS wobjdata WObj \switch
     Corr)
example1_count:=example1_count + 1;

  CapLTopoint, Speed, IntCdata, IntWeavestart, IntWeave, Zone, Tool
       \wobj?wobj;
ERROR
  ResetIoSignals;

  IF no_of_retries > 0 THEN
    IF err_cnt < no_of_retries THEN
      err_cnt := err_cnt + 1;
      Skipwarn;
      err_code := new_aw_errMsg();
      RETRY;
    ELSE
      err_cnt := 0;
      Skipwarn;
      err_code := new_aw_errMsg();
      RaiseToUser \Continue \ErrorNumber:=err_code;
    ENDIF
  ELSE
    Skipwarn;
    err_code := new_aw_errMsg();
    RaiseToUser \Continue \ErrorNumber:=err_code;
  ENDIF
ENDPROC

FUNC errnum new_aw_errMsg (\switch W)
  VAR errnum ret_code;
TEST ERRNO
  CASE CAP_PRE_ERR:
    ! Check of signals here
    ret_code := AW_EQIP_ERR;
  ENDTEST
  RETURN ret_code;
ENDFUNC
ENDMODULE
```

The `errnum` raised to the level above arcl_move_only is AW_EQIP_ERR, that is, the CAP error CAP_PRE_ERR is replaced by AW_EQIP_ERR and the CAP error will not appear in the error log (topic *Process*).

2.9.2 Writing a general error handler

General error handlers

For a CAP single system you can write an error handler as shown in examples 1 and 2. For a MultiMove system, errors must be handled as in example 3 below.

The error handlers for all tasks executing synchronized motion must contain the following:

For recoverable errors handled application internally

```
StorePath;
! Here you may have some application specific actions/movements
RestoPath;
StartMoveRETRY;
```

For errors handled by the user:

```
RaiseToUser \Continue \ErrorNumber:=error_code;
```

Not all instructions must be in all tasks, but some are important. `RETRY` must be changed to `StartMoveRETRY` in all RAPID tasks if running in synchronized mode. `StartMoveRETRY` restarts all path processes. For a task using the instruction `MoveExtJ` or other move instructions like `TriggX` or `MoveX` the `StartMoveRETRY` orders that mechanical unit to start again. The `RETRY` order works only on the `CapL`/`CapC` instructions.

Example 3

This example only shows the error handler.

```
IF err_cnt < no_of_retries THEN
  err_cnt := err_cnt + 1;
  ! Suppress CAP error message
  Skipwarn;
  ! Remap CAP error to application specific error
  err_code := new_aw_errMsg();
  ! Store current movement information
  StorePath;

  ! Here you may have some application specific actions/movements

  ! Restore movement information stored with previous StorePath
  RestoPath;
  ! Restart the movement
  StartMoveRETRY;
ELSE
  err_cnt := 0;
  ! Suppress CAP error message
  Skipwarn;
  ! Remap CAP error to application specific error
  err_code := new_aw_errMsg();
  ! Raise the application specific error to the RAPID user level
  RaiseToUser \Continue \ErrorNumber:=err_code;!***
ENDIF
```

Continues on next page

2.9.2 Writing a general error handler
Continued

> **Note**
>
> `StartMoveRETRY` is an instruction that combines `StartMove` and `RETRY`. A `StartMove` executed on `TriggX`, `MoveX` or `MoveExtJ` will order a restart of movement for that specific robot/additional axis. It is the `RETRY` that triggers CAP process to restart and order a restart of movement.

RaiseToUser

`RaiseToUser` can be used with `\Continue` or `\Breakoff`.

- `\Continue` will raise the error to user level (first level that is not in a NOVIEW module). A `RETRY` from that level will start the instruction that failed exactly where the error occurred - that is, it will not re-run the NOVIEW procedure as a whole.
- \Breakeoff will rerun the instruction where the program pointer is on user level.

If `RaiseToUser` is used, the user itself has to restart or enable restart of robot movement by using:

- `StartMove` or `StartMoveRETRY` will restart movement as soon as all synchronized robots have sent their restart orders.
- `StartMoveReset` enables manual restart (start button on the FlexPendant).

2.10 Restart

Description

If the execution of a CapL/CapC instruction is stopped due to an recoverable error or a program stop, a backing distance can be specified at restart of the instruction. The backing distance is a value the user must specify in the capdata data structure (see capdata in *Technical reference manual - RAPID Instructions, Functions and Data types*).

Sensors

Sensors can be used together with CAP. The different sensor systems implemented are At-PointTracker (for example, serial *WeldGuide*) and Look-Ahead-Tracker (for example, laser tracker). See also *Application manual - Controller software IRC5* as well as CapL CapC and captrackdata in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Units

In CAP the following units are used:

length	mm
time	s
speed	mm/s
angle	degree

Tuning

It is possible to change the value of (tune) the following data when it is active, during execution:

weavedata components:

- width
- height
- bias

capdata components:

- speeddata

Example

The example changes the main speed and weave within a TRAP.

```
VAR intnum intno0;

PROC main()
  IDelete intno0;
  CONNECT intno0 WITH MainMotionTrp;
  ICap intno0, MAIN_MOTION;
  CapL p11, v100, cdata1, weavestart, weave, z20, tool0;
ENDPROC
```

Continues on next page

```
TRAP MainMotionTrp
  cdata1.speed_data.main := 23;
  weave.width := 5;
ENDTRAP
```

Note

In this example the TRAP-routine is inside the main module. The recommendation is that all TRAP-routines should be executed by a background task.

2.11 System event routines

Introduction

CAP knows nothing about external equipment so the control of the equipment must be handled in 'shelf-hooks' (stop-, start-, restart-, ...) or in `ICap` trap events, see `ICap` in *Technical reference manual - RAPID Instructions, Functions and Data types*. Normally the `ICap` trap events are used to activate and deactivate equipment.

All errors and stops generated directly from the CAP source code in the controller software, can be handled in the `CAP_STOP ICap` event. The `CAP_STOP` trap event is executed as soon as CAP detects an error (fatal or recoverable) or if the RAPID program is stopped with an active CAP process, that is, a `CapX` instruction has been executed, but no `CapX` instruction with `last_instruction:=TRUE` (see `capdata` in *Technical reference manual - RAPID Instructions, Functions and Data types*). The procedure that should be run when a stop occurs must deactivate external equipment. The stop shelf is necessary to take the system to a fail-safe state, if anything unexpected happens in the controller software.

Exceptions

Not all errors can be handled in shelf-hooks or with the `CAP_STOP` trap. If the system, for some reason, is forced to system failure state, all execution of RAPID code is immediately stopped. To handle this situation, the signal definitions (in EIO.cfg) for signals that handle external equipment must contain the flag `-SysfailReset`, which will set the signal to its default value (0 or defined by the flag `-default`). We recommend defining those signals with `CapCondSetDO` to be set to a defined status, when TCP movement stops, see `CapCondSetDO` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

2.12 Limitations

Limitations

- Problems can occur if much logging to files is added in CAP and user event trap routines. The reason is that file writing takes long time and will delay the execution of the next instruction. That may cause corner path failure, stopping the movement of the robot for a short time, which may be fatal for the process (for example, arc welding).
- CAP does not support error recovery with long jump.

3 Programming examples

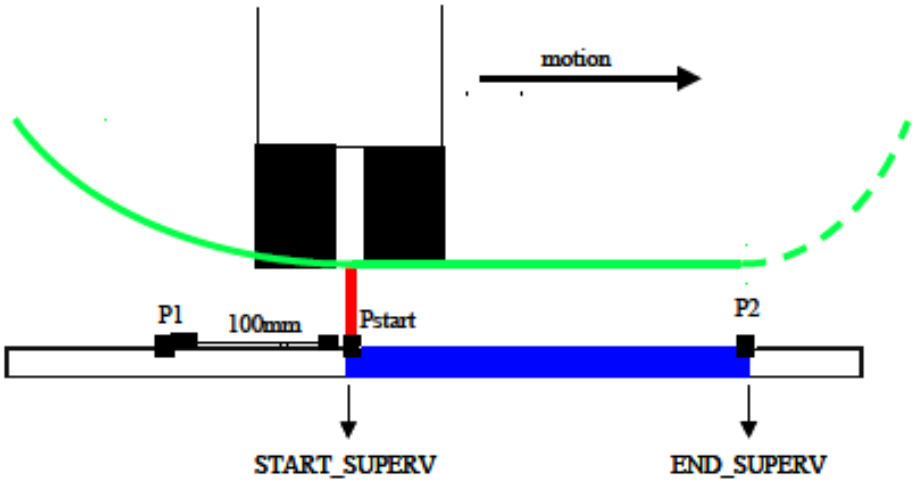
3.1 Laser cutting example

Requirements

- a slot is to be cut into a number of metal sheets with a laser
- at the starting point of the slot accuracy is not critical
- at the finishing point of the slot accuracy is required
- the application is time critical, that is, it has to be as fast as possible

CAP setup to meet the requirements

- *flying start*: the robot can move with speed past the start point (P1) and start the process on the fly between point P1 and Pstart.
- *normal end*: the robot must cut all the way to the end point (P2) and stop before turning off the laser and moving on to next cycle.
- In order to assure the quality of the cuts the process needs to be started at the latest one second after passing Pstart. Three seconds are given for ending the process.



xx1200000172

3.2 Step by step

Set up CAP events

First there is a need to set up the events that are going to be ordered from CAP. For this application a minimum of two events are needed: START_SUPERV given at the Pstart point for starting the process. END_SUPERV given at the end point for turning off the process. That is done the following way:

```
VAR intnum start_intno:=0;
VAR intnum end_intno:=1;

TRAP start_trap
  SetDo doLaserOn, high;
ENDTRAP

TRAP end_trap
  SetDo doLaserOn, low;
ENDTRAP

IDelete start_intno;
IDelete end_intno;
CONNECT start_intno WITH start_trap;
CONNECT end_intno WITH end_trap;
ICap start_intno, START_MAIN;
ICap end_intno, END_MAIN;
```

Set up supervision

In this case only one signal, *diLaserOn*, needs to be supervised, but in three different process phases:

1. *diLaserOn* goes high (ACT) in the START phase.
2. *diLaserOn* stays high (that is, triggers on PAS) during the MAIN phase.
3. *diLaserOn* goes low (PAS) in the END phase.

Furthermore we need to setup the start point for supervision of the START phase, and setup the time-out timers for the START and the END phase.

```
SetupSuperv diLaserOn, ACT, SUPERV_START;
SetupSuperv diLaserOn, ACT, SUPERV_MAIN;
SetupSuperv diLaserOn, PAS, SUPERV_END_MAIN;

capdata.start_fly_point.process_dist := 0;
capdata.start_fly_point.distance := 100;
capdata.sup_timeouts.start_cond := 1;
capdata.end_fly_point.process_dist := 0;
capdata.end_fly_point.distance := 0;
capdata.sup_timeouts.end_main_cond := 3;
```

Continues on next page

The main program

The user might use an encapsulation of `CapL` and call it for example `CutL`, which might be defined as follows:

```
PROC CUTL (...)
  MoveL p1, v100, z10,...
  CapL p2, v100, cdata, startweave, weave, fine, tool0, ...
  MoveL px, ...
ENDPROC
```

This page is intentionally left blank

4 RAPID reference

4.1 Overview

About the RAPID components

This is an overview of all instructions, functions, and data types in *Continuous Application Platform*.

For more information, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instructions

Instructions	Description
CapAPTrSetup	CapAPTrSetup (*Setup an At-Point-Tracker*) is used to setup an At-Point-Tracker type of sensor, for example, *WeldGuide* or AWC.
CapC	CapC is used to move the tool center point (TCP) along a circular path to a given destination and at the same time control a continuous process.
CapCondSetDO	CapCondSetDO is used to define a digital output signal and its value, which will be set when the TCP of the robot that runs CAP, stops moving during a CAP instruction (CapL or CapC) before the CAP sequence is finished.
CapEquiDist	CapEquiDist is used to tell CAP to generate an equidistant RAPID event (EQUIDIST) on the CAP path. The first event is generated at the startpoint of the first CAP instruction in a sequence of CAP instructions. From RAPID it is possible to subscribe this event using ICap.
CapL	CapL is used to move the tool center point (TCP) linearly to a given destination and at the same time control a continuous process.
CapLATrSetup	CapLATrSetup (*Set up a Look-Ahead-Tracker*) is used to set up a Look-Ahead-Tracker type of sensor, for example, Laser Tracker.
CapNoProcess	CapNoProcess is used to run CAP a certain distance without process.
CapRefresh	CapRefresh is used to tell the CAP process to refresh its process data. It can for example, be used to tune CAP process parameters during program execution.
CapWeaveSync	CapWeaveSync is used to setup weaving synchronization signals without sensors. The I/O signals must be defined in EIO.cfg.
ICap	ICap is used to connect an interrupt number (which is already connected to a trap routine) with a specific CAP Event, see Arguments below for a listing of available Events. When using ICap an association between a specific process event and a user defined Trap routine is created. In other words, the Trap routine in question is executed when the associated CAP event occurs.
InitSuperv	InitSuperv is used to initiate CAP supervision. This means that all supervision lists will be cleared and all I/O subscriptions will be removed.
IPathPos	IPathPos is used to retrieve the position of the center line during weaving.
RemoveSuperv	RemoveSuperv is used to remove conditions added by SetupSuperv from supervision.

Continues on next page

Instructions	Description
SetupSuperv	SetupSuperv is used to set up conditions for I/O signals to be supervised. The conditions are collected in different lists:

Functions

Functions	Description
CapGetFailSigs	CapGetFailSigs is used to return the names on the signal or signals that failed during supervision.

Data types

Data types	Description
capdata	capdata contains all data necessary for defining the behavior of the CAP process.
caplatrackdata	caplatrackdata contains data, with which the user can influence how the CapL/CapC instructions incorporate the path correction data generated by a Look-Ahead-Tracker (for example, Laser Tracker). caplatrackdata is part of the captrackdata.
capspeeddata	capspeeddata is used to define all data concerning velocity for a CAP process - it is part of capdata and defines all velocity data and process times needed for a CAP process:
captrackdata	captrackdata provides the CapL/CapC instructions with all data necessary for path correction with a Look-Ahead- or At-Point-Tracker. The data is passed to the CapL/C instructions with of the optional argument \Track.
capweavedata	capweavedata is used to define weaving for a CAP process during its MAIN phase (see *Supervision and process phases on page 22*).
flypointdata	flypointdata is used to define all data of flying start or flying end for a CAP process - it is part of capdata for both flying start and flying end.
processtimes	processtimes is used to define the duration times for all status supervision phases in CAP, except phase MAIN, which is defined by the robot movement (see *Supervision on page 18*).
restartblkdata	restartblkdata is used to define the behavior of a CAP process at restart.
supervtimeouts	supervtimeouts is used to define timeout times for handshake supervision in CAP.
weavestartdata	weavestartdata is used to control stationary weaving during start and restart of a process in CAP.

Index

Contact us

ABB AB, Robotics
Robotics and Motion
S-721 68 VÄSTERÅS, Sweden
Telephone +46 (0) 21 344 400

ABB AS, Robotics
Robotics and Motion
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.
Robotics and Motion
No. 4528 Kangxin Highway
PuDong District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

ABB Inc.
Robotics and Motion
1250 Brown Road
Auburn Hills, MI 48326
USA
Telephone: +1 248 391 9000

www.abb.com/robotics

3HAC050990-001, Rev C, en