ABB Robotics

Application manual
RAPID development guidelines for handling applications

Power and productivity
for a better world™

Trace back information:
Workspace Main version a23 (not checked in)
Published 2013-02-01 at 10:15:54
Skribenta version 1184

Application manual
RAPID development guidelines for handling applications

Document ID: 3HAC046417-001
Revision: -

The information in this manual is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this manual.

Except as may be expressly stated anywhere in this manual, nothing herein shall be construed as any kind of guarantee or warranty by ABB for losses, damages to persons or property, fitness for a specific purpose or the like.

In no event shall ABB be liable for incidental or consequential damages arising from use of this manual and products described herein.

Additional copies of this manual may be obtained from ABB.

The original language for this publication is English. Any other languages that are supplied have been translated from English.

ABB AB
Robotics Products
SE-721 68 Västerås

Table of contents

This page is intentionally left blank

Overview of this manual

About this manual

This manual explains programming guidelines while handling applications under RAPID software development.

Who should read this manual?

This manual is primarily intended for experienced programmers.

Prerequisites

The reader should be well versed in

- industrial robots and their basic terminology,
- the Rapid programming language
- and with the system parameters and their configuration.

References

References	Document ID
Program design guidelines - Part B / S4 Handbook of Methods	

Revisions

Revision	Description
-	First edition, Robotware 5.15.

License agreement

License agreement for RobotWare Machine Tending

1 ABB is the only owner of the copyright and usage rights in the software option RobotWare Machine Tending that is delivered.

2 ABB assigns to the licensee a simple, non-transferable, exclusive, but unlimited right to use the option RobotWare Machine Tending.

3 The license entitles the user only to the "proper use" of the software option RobotWare Machine Tending on a robot controller. The licensee is not allowed to replicate the option RobotWare Machine Tending or parts of it and make these accessible to third parties or the use the software or parts of it on other robot controls. Taking a back-up copy exclusively for own use on the original hardware is exempted from this.

4 Modifying, translating, reverse engineering or decompiling or disassembling the software option RobotWare Machine Tending is not allowed.

Product documentation, M2004

Categories for manipulator documentation

The manipulator documentation 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 M2004 manipulator systems.

Product manuals

Manipulators, controllers, DressPack/SpotPack, and most other hardware will be 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.

Application manuals

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

Continues on next page

Continued

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 - Introduction to RAPID*
- *Operating manual - IRC5 with FlexPendant*
- *Operating manual - RobotStudio*
- *Operating manual - Trouble shooting IRC5*, for the controller and manipulator.

Safety

Safety of personnel

A robot is heavy and extremely powerful regardless of its speed. A pause or long stop in movement can be followed by a fast hazardous movement. Even if a pattern of movement is predicted, a change in operation can be triggered by an external signal resulting in an unexpected movement.

Therefore, it is important that all safety regulations are followed when entering safeguarded space.

Safety regulations

Before beginning work with the robot, make sure you are familiar with the safety regulations described in the manual *Operating manual - General safety information*.

This page is intentionally left blank

1 Introduction

1.1 General

The RAPID programming language assumes knowledge in the fundamentals of higher level programming languages. This manual does not provide basic information about RAPID programming and the available instructions and functions.

Programming styles, as we know, are different and each one who ever tried to read and understand a program which has been written by someone else knows, how frustrating the result can be.

The purpose of this manual is to provide a programming standard for handling applications. It covers the experience of more than 15 years of software development in this field.

This standard does not only mean the program code itself but also includes the steps for preparation as well as the program documentation

Robot programmers of ABB, subcontractors and others should make use of these guidelines, with the goal to have a common open standard for RAPID programs. This will help to make different handling programs more comprehensible for programmers, technicians and customers.

The program elements in this manual are named in accordance with the guidelines compiled by ABB: "Program design guidelines - Part B / S4 Handbook of Methods".

Note

Users of the ABB software option Machine Tending Solution (MTS) note that this manual does not describe special functionalities of MTS. Nevertheless if a RAPID application is programmed by considering these guidelines, it will be easier to use MTS. If the HomeRun functionality of MTS shall be used in a software project, these guidelines are binding. Otherwise, HomeRun will not work properly.

This page is intentionally left blank

2 Project flow

2.1 Project planning procedure

Before writing the robot program and using these guidelines in the actual sense, the programmer should first of all understand the task description and put it down in a form that is meaningful to him.

The following steps must be complied with for this purpose:

1 Preparing a process flow description of the entire system.

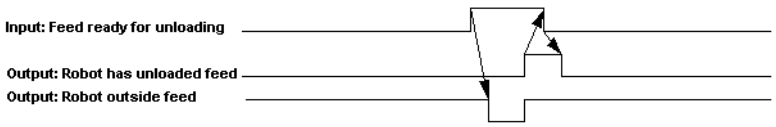
To begin with, the programmer should be clear about what needs to be programmed. A process flow description of the system is indispensable for this purpose. This process flow description (flow chart), however, should not only contain the normal execution, but also the strategies for abnormal conditions (error handling).

2 Creating a position overview.

A position overview facilitates quick detection of the stations (for example, machines, conveyors, slides…) that are located in the system and is helpful in finding your way in the process flow description and for naming or designating the robot positions associated with the stations.

3 Defining the required signals.

The signals can be defined once the stations located in the system are known and how they communicate with one another. In the process, it should be noted that a "High active handshake" must always be used for the communication. A non-existent OK signal is not to be evaluated automatically as an OK indication, since on account of wire breakage or defective sensors it cannot be ensured that the signal is always working properly (example: Instead of an input signal meaning "Production without the robot", a signal with the meaning "Production with the robot" should be used. This rules out inadvertent movements of the robot in the event of wire breakage). Release signals for an action by the robot as well as acknowledgments should follow the following example (Signal flow for unloading a feed by the robot):



en1300000185

When communicating with a PLC, only the edge of a signal from the PLC must be evaluated (for example, for release signals). This is so because a signal present at the robot must not necessarily also be a clearance signal (Robot controller = Down-stream controller).

Only those signals with which the robot sends indications that it is beyond the disturbance range of a machine, etc., should be evaluated statically.

All signals to the robot (Outputs of the PLC) are static signals, which must be acknowledged accordingly by the robot. Sending pulse signals is not permissible.

Continues on next page

Continued

4 Documenting the signals.

   After knowing the signals that are used, these must also be documented. The documentation of the signals always includes the name, a brief explanation and a detailed explanation. The brief explanation serves as an enumeration of the name, and the detailed explanation is an enumeration of the function of the signal. A sample of such documentation is provided in the *Sample program on page 45*.

5 PLC interfacing / defining and documenting handshaking.

   In order to explain the function of the individual signals even better, a "Signal step diagram" should be prepared to illustrate the chronological flow of the signals and, thus, to clarify the function of the signals. An example of such a signal step diagram is also provided in the sample documentation.

   The process flow description, the documentation of the signals and the signal step diagram must be coordinated with the PLC programmers.

6 Specifying error handling.

   The following aspects must be clarified:

   - What errors or faults can occur?
   - How should the operator respond?
   - How should the robot respond?

7 Specifying password assignment.

   Should passwords be assigned, what should they be and which levels should be password-protected?

8 Defining task distribution for the background tasks (only for multitasking).

9 Writing the program (including backgound tasks).

10 Documenting the program.

3 Program structure

3.1 Process flow diagram

A process flow diagram must always be prepared at the beginning of the project. If this important preparatory work is not done, it almost always leads to problems, particularly with complex programs.

The preparation of a process flow diagram compels the programmer to work in a structured manner and enables critical parts of the project to be detected in the initial stage. This avoids undesirable surprises when the project is already in an advanced stage of implementation. Moreover, a process flow diagram simplifies and facilitates induction into the project in the event of any necessary changes in the team.

The resources for the preparation of a process flow diagram are for example:

- Flow chart
- Nassi-Shneidermann diagram
- Simplified system layout with position overview
- Process flow description / System description

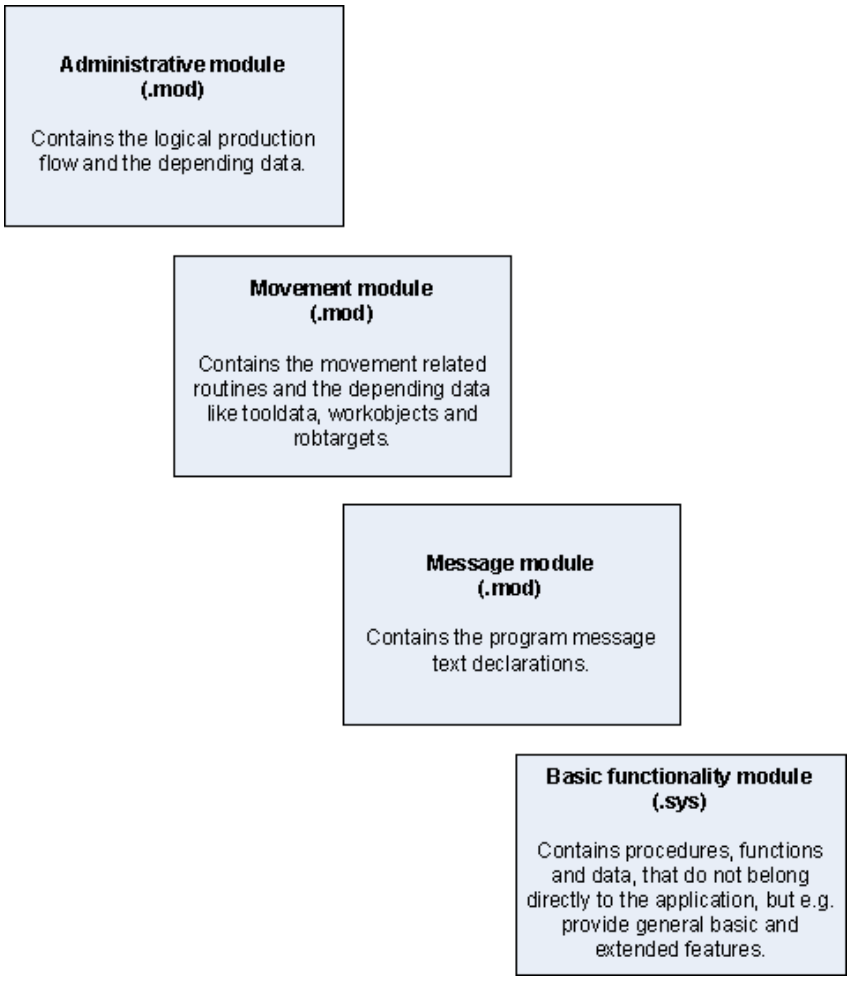
3.2 Program structure

3.2.1 Introduction to the program structure

In general, the following applies to the RAPID program structure:

- Strict demarcation between movement routines and administrative routines.
- BASE.SYS and USER.SYS remain unchanged.
- Standard utilities are saved in separate system modules and are possibly encoded. Such utilities may be: Palletizing modules, data transfer to visualization systems, special self instructions, and so on. These standard routines, if required, must be adapted to the year of manufacture.
- Data for processes can be saved in a separate program module (for example, PRGPARAM.MOD).

The following figure illustrates how the basic structure of a RAPID program for handling applications should look like.



en1300000181

3.2.2 System modules

At the time of delivery, the system modules "BASE" and "USER" are already available as the operating system in the controller. These should not be changed.

System-specific routines and data are saved in their own system modules, which are protected from external attack using special attributes (for example, NOVIEW or encoding of the source code).

3.2.3 Program modules

The *Sample program on page 45* comprises at least of three modules:

Module	Description
<CellName>.mod	Main program module, which contains the procedure main() and the general production flow.
Movement.mod	Movement module with movement routines and the related data.
Message.mod	Module with messages and text fragments.

Moreover, it is possible to add other modules containing data and routines to the programs, with these modules being applicable to several robot stations (for example, tools (TCPs), work-piece objects, etc.), or containing different setup routines. These should be created as a program module so that they can be saved as a separate module and may be loaded in another robot controller.

The division has been selected on grounds of practicable handling. Thus, for example, messages are saved only in the module `Message.mod` and may, if required, be completely revised or translated in a foreign language.

When saving the program, all program modules (*.MOD) are saved in a common directory. In addition, a program file ending with `.PGF` is generated and this ensures that all modules saved can be loaded again jointly.

Note

When loading a program `< CellName >.pgf`, all program modules `*.MOD` already loaded are deleted or overwritten with the new modules. In contrast, the system modules (`*.SYS`) must be loaded or deleted separately.

3.2.4 System parameters

The system parameters are stored in a separate directory called SYSPAR on the data medium. Basically, the system parameters from the controller are saved in this directory.

3.2.5 Backup or restore

The backup feature saves all system parameters, system modules and program modules in a single operation. The data is stored in a directory that the user must specify.

In addition to the backup, the program should always also be saved separately.

The restore feature restores the data from a backup directory. Restore replaces all system parameters and loads all modules from the backup directory. A warm restart is initiated after restoring the data.

3.3 Naming the data

3.3.1 General naming conventions

The "Handbook of Methods" is binding for the assignment of names. The prefixes defined in this handbook for individual variables, constants and persistents, in fact, reduce the number of characters available for choosing the rest of the name, but however, they are indispensable for clear and unique identification. In a sample query, such as `IF Start = 1 THEN...`, for example, it cannot be concluded whether what is being referred to is an input or something else. It is only with the prefix `di` that it becomes clear that what is being referred to here is a digital input (`IF diStart = 1 THEN...`).

3.3.2 Convention on the position names

The following specifications are laid down as an extension to those given in the "Handbook of Methods":

- Position names basically begin with the letter "p".
- Special default or standard positions have identifying names (`pHome`, `pReady`, etc.)
- All other positions are named according to the following convention:

Format	Description
pXX(X)	**p**:Position prefix
pXX(X)_TY	**XX(X)**: 2-digit (or 3-digit) station name beginning with 10 (100) (e.g.: Separation (10 or 100), Oiling (20 or 200), etc..) **T**:Type prefix for index according to different part types **Y**:Type index according to different part types

Examples:

```
p10, Preliminary position at station 10
p11, Pick position at station 10
p12, End position at station 10

p20 Preliminary position at station 20
…

p20_T1 Preliminary position at station 20 for part type 1
p21_T1 Drop position at station 20 for part type 1
p22_T1 End position at station 20 for part type 1
```

A station (machine) can have any number and there is no mandatory specification. The only condition: The meaning of a number must be explained on a position overview diagram.

For subordinate (or secondary) intermediate positions, serial numbering or the selection of star "`*`" positions has been a proven method for the `robtargets`, since it is impossible to find an intelligent and self-explanatory name for each position and one that still fits within the maximum length of a position name.

A sequence of * positions (even individually) should always be enclosed by positions with clear and unique names, for example, `p10`, `*`,..., `p11`.

The home position should always be the position `p99` or `p999`.

3.3.3 Naming the movement routines

The names of movement routines always identify the starting and ending point of a movement.

In general, all movement routine names begin with the prefix `mv`.

`mvStartpoint_Endpoint`

Example

mv10_999	moves the robot from the station 10 to the home position.
mv10_20	moves the robot from station 10 to station 20.

As a rule, movement to the first position (starting position) takes place only in programming mode and at slow speed. In this manner, uncontrolled movement from the end (final) position back to the starting position while testing the robot movements is prevented.

It cannot be prevented that a worker operates a movement routine in continuous mode, which means that after the last instruction in the routine it starts executing again from the fist instruction onwards. Slow speed may therefore prevent uncontrolled movement and major damage.

Example

```
PROC mv999_10 ()
!Von : Homeposition
!Nach: Prepos. station 10
IF OpMode()<>OP_AUTO MoveJ p999,v200,z10,tGripper;
MoveJ *,v2500,z10,tGripper;
MoveJ p10,v2500,z10,tGripper;
ENDPROC
```

3.3.4 Use of type-dependent movement routines

If different types of work-pieces need to be moved using different movement rou-tines in a robot program, but are handled with the same program execution flow, these are identified by a type-dependent index, which is also provided with the "T" prefix.

Example:

Movement from position 10 to 20 for type number 3 with the type prefix:

Routine name: `mv10_20_T3`

The advantage of this method lies in the fact that, in general, only the movement routines change, but not the general program execution flow.

As a result, the administrative routines that invoke the movement routines must not be rewritten for each type, but instead, only the index for the routine call needs to be adapted. This takes place preferably with the help of routine calls with late binding (Late Binding "`%string%`").

Example:

```
%"mv10_20_T"+ValToStr(nTypeNo)%; (with type prefix)
```

or with

```
CallByVar "mv10_20_T",nTypeNo;
```

Benefit:

The administrative program needs to be modified only once in case of any changes in its execution flow.

> **Note**
>
> When using type-dependent routines, the position names are also indexed.

Example:

```
PROC mv12_20_T1()
!From: End position station 10
!To: Preliminary position station 20
FirstMove\J,p12_T1,v200,z10,tGripper;
MoveJ *,v2500,z10,tGripper;
MoveJ p20_T1,v2500,z10,tGripper;
ENDPROC
```

3.3.5 Naming the I/O signals

- For the identification of the inputs and outputs, the codes di/do for digital inputs and outputs, ai/ao for analog inputs and outputs and gi/go for group inputs and outputs must be used (for example, `diProgStop`, `doMotorOff`).
- The name of the signal should be chosen to be as self-explanatory as possible and should be related to the "1" status of the function (for example, `diGripperOpen` for "Gripper is open").
- Any assignment of the physical channels within the signal names must be avoided (for example, `di1_CycleEnd`).
- The somewhat favored naming convention like `di1`, `do5`, ..., which indicates the channel where a signal is located, may be somewhat helpful while testing the hardware, but does not serve to improve the readability of the programs. It is not so important where a signal is present, but instead, that it is present.
- The position, name (or designation) and explanation of the signals must be documented in any case.

3.4 Error handling

As a rule, all programs should be written so that they are as reliable as possible. This means that every possible error or fault condition should be trapped by the program on its own in order to enable defined error handling and to thereby ensure streamlined and trouble-free operation of the system as far as possible. The program documentation should describe the possible errors and how they are handled. Any response to an error condition that occurs definitely depends on the situation. For this reason, standard routines should always have their own error handlers, which treat the possible error regardless of the procedure / function in which the error occurs. In general, it would be possible to have a standard error handler in the main routine in order to be able to trap the error at least globally.

Furthermore, it is possible to implement the strategies necessary for conditions of faults or failures with the help of error handling, by invoking separate error numbers (1-90) using the `RAISE` instruction. In this process, the program exits the routine with a specific abort or cancelation, and continuation of program execution is enabled via the error handling routine with suitable measures and actions.

Example:

```
CONST errnum erToHomePos:=89;

!*********************************************************
!* Procedure Example()
!*
!* Description:
!*
!* Example of error handling
!*
!* Date: Version: Programmer: Reason:
!* 2013-01-01. 1.0 Mr. Example created
!*********************************************************
PROC Example()
!Release or cancelation is expected
WaitUntil diRelease=high or diIRBtoHome=high;
!With the request move "IRB to the home position"
!the program is terminated via error handling
IF diIRBtoHome=high RAISE erToHomePos;
…
ERROR
IF ERRNO=erToHomePos THEN
mv100_999;
RAISE;
ENDIF
ENDPROC
```

In the example illustrated above, the procedure `Example` waits for the input `diRelease` or the cancelation signal (for example, `diIRBtoHome`). If the cancel signal is set, the error handling routine is invoked with the error number `erToHomePos`. The error handling routine for this error number in the example

Continues on next page

Continued

given contains a movement of the robot (`mv100_999`) and a call to the error handling system of the previous routine (`RAISE`). The error should be handled appropriately in this previous routine.

3.5 The program structure

The basic structure of the main program module should not be modified in order to enable similar appearance of the routines and to be able to find one's way in the program. A *Sample program on page 45* has been furnished in a later chapter.

The main routine could have the following structure:

```
PROC main()
!Initialize the start values
Init;
!Check if the gripper is working
CheckGripper;
!Move to the preposition of the first station
Mv999_100;
!Repeat until automatic mode is de-selected
WHILE diAutomatic=high DO
!Output program information
ProgInfo;
!Execute one production cycle
Production;
ENDWHILE
!Move to home position
Mv100_999;
STOP;
ENDPROC
```

The predefined routines must "merely" be provided with their contents.

3.6 Write-protection of the modules / password assignment

After the station has been commissioned, all modules may be assigned the `VIEWONLY` attribute so that any unprotected manipulation by the worker is ruled out. The `VIEWONLY` attribute can only be changed on one PC.

Moreover, the robot can be provided with passwords at various levels so that no un-authorized access to the robot is possible. For this reason, passwords must be assigned at any cost and controlled.

Note

No password is specified by default. For this reason, the operator can initially make any modifications.

This page is intentionally left blank

4 Program documentation

4.1 Introduction

In order to ensure proper program creation and maintenance, programs must be docu-mented in accordance with the following aspects:

1. A complete program printout with all procedures, functions and routines (other than system modules with standard routines) must be created.
2. With the help of a simple, clearly structured and commented program structure it must be ensured that detailed program flow charts of all procedures and routines may be dispensed with.

   This is ensured to a large extent by creating the routine `main()` and the routines called within it. For station-specific complex or cluttered program flows, structure charts or flow charts (for example with MS Visio) are prepared. These should, however, not represent any 1:1 implementation of the RAPID program, but instead, it should have an overview of all basic logical flows for the contents.
3. The functional descriptions of the standard routines used may be incorporated into the documentation for the sake of completion. System-specific processing is omitted.
4. All robot positions (starting and end positions of a movement) must be represented as position numbers in a graphical image (for example simplified system layout). This can, however, be limited to the station number if the starting and end positions required are described in a separate list.
5. All signals are described in a signal list with names, meanings, channel numbers, etc.
6. For all other system parameters, the description for parameter modification must be given separately in the documentation.
7. If needed, other supplementary documents may be added.

4.2 Structure for the creation of the program documentation

1 Program Structure
   a General Rules
   b System Parameters
   c System Modules
   d Program Modules
   e Naming Conventions
      I General Naming Conventions
      II Naming of Positions
      III Naming of Movement Routines
      IV Naming of Digital Inputs and Outputs
2 Program Description
   a Position Layout
   b Position Description
   c Description of the Production Cell
      I Components of the Production Cell
      II Description of the Production Flow
      III Production Flow in Case of Errors
   d Flowchart of the Program Flow
3 Overview of the Progam- and System Modules
4 Guidance for Commissioning
   a Setting up the Worldzones and Event Routines
   b Multitasking
      I Available Tasks
      II Registering the Tasks
      III Preparing the Tasks to be Loaded Automatically
      IV Loading the Task Modules into the Memory

4.3 Headers and program information

4.3.1 Program header

A standard module header like the one shown below can be used. It needs to be filled up once at the time of setting up the project and it must be then updated in case of program changes:

```
!***********************************************************************
!*
!* Station : System Name (Designation)
!*
!***********************************************************************
!*
!* Customer : Customer company name and location
!*
!* Project name : Project Description
!* Project no. : Order Number
!*
!* Robot no. : IRB ??-?????
!* Software : RobotWare 5.15
!* Revision : 0
!* Key Contr. : ThEgOOdDiEyOuNg
!* Key Drive : AvEcAeSARmoRiTuriTEsAlUTaNT
!*
!* Author : Name of the author
!* Company : The company name,
!* City / State
!* Department : Name of department
!* Telephone : +67(0)12345/99 – 0
!*
!* Hotline : +67(0)12345/99 – 1
!*
!* Version : 1.0
!* Created : 2013-01-01
!*
!* Modification Date: Name: Reason:
!* 2013-02-31 Mr. Example New release signal
!*
!***********************************************************************
```

Additional information may be provided such as the contents of the module and the software version of the routines incorporated.

4.3.2 Module header

The module header for program and system modules looks like the following:

```
!*********************************************************
!*
!* Module name: Name of the module
!*
!*********************************************************
!*
!* Description:
!*
!* General description of the module functionality
!*
!*
!* Date: Version: Programmer: Reason:
!* 2013-01-01 1.0 Mr. Example created
!*********************************************************
```

4.3.3 Routine header for procedures and functions in the user program

Administrative routines have a routine header, which describes the function of the routine. Program modifications or adaptations in the routine must be marked in the program header with the date, programmer and reason for the same.

Example:

```
!*********************************************************
!*
!* Sample procedure
!*
!* Description:
!*
!* General description of the routine function
!*
!*
!* Date: Version: Programmer: Reason:
!* 2013-01-01 1.0 Mr. Example created
!* 2013-01-02 1.1 Mr. Example bTest added
!*********************************************************
```

A detailed routine header is definitely not necessary for movement programs since a movement program should not contain anything other than move commands. Hence, it is not mandatory to provide a routine header in such cases.

Example:

```
PROC mv999_100()
!From: Home position
!To : Prelim. pos. DGM
IF OpMode()<>OP_AUTO MoveJ p999,v200,z10,tGripper;
MoveJ p100,v2500,z10,tGripper;
ENDPROC
```

4.3.4 Routine header for standard procedures and functions

Standard routines contain a routine header, which gives an explanation of the routine as well as the data used. In the process, the transfer parameters, the return parameters as well as all the data that the routine modifies or needs are listed and described. Lines that are not needed in the headers may be omitted.

In addition, an example is added from which it becomes clear how the routine is to be used or how its parameters are to be configured.

Example:

```
!*********************************************************
!*
!* ROUTINE NAME: <name of the routine>
!*
!*********************************************************
!*
!* DESCRIPTION: <what the procedure does (not how)>
!*
!* IN: <parameter name and description>
!*
!* OPTIONAL: <parameter name and description>
!*
!* INOUT: <parameter name and description>
!*
!* RETURN: <parameter type>
!*
!* ASSUMPTIONS: <list of each external variable,
!* control, open file, or other element
!* that is not obvious>
!*
!* EFFECTS: <list of each affected external variable
!* control or file and the effect it has
!* (only if this is not obvious) >
!*
!* NOTE: <internal remarks (only if this is not
!* obvious) >
!*
!* EXAMPLE: <example, how the routine is used
!* (only if this is not obvious)>
!*
!*
!* Date: Version: Programmer: Reason:
!* 2013-01-01 1.0 Mr. Example created
!*********************************************************
FUNC datatype RoutineName(Arguments)

ENDFUNC
```

4.3.5 Program information

Each program must contain information about its origin and subsequent modifications. Moreover, the tasks and the process flows of the system may be described in a special module.

It should contain the following information as a minimum:

- Station name
- Current program version
- Name of the programmer
- Serial number of the robot
- Hotline number, if this is different from the ABB hotline
- Note on the info routine, if available

This information can either be in the form of comments at the beginning of the routine `Main` or it may be displayed via a separate routine on the FlexPendant.

Example:

```
!*********************************************************
!* Procedure main *
!* *
!* Description: *
!* *
!* This procedure sets the initial values for execution *
!* *
!* Date: Version: Programmer: Reason: *
!* 2013-01-01 1.0 Author created *
!*********************************************************
PROC main()
!*****************************************
!System data
!Customer : Company XYZ
!System : DCM Operation
!SN no. : 24-12345
!Author : Mr. Example
!Date : 2012-01-01
!Hotline : +67(0)1234/56-798
!*****************************************
!
....
ENDPROC
```

4.3.6 Disturbance range signals

In addition to the normal handshake for program execution, in case of machine operations, the robot should also send a signal to the corresponding machine that it is not located in its disturbance range. Safe working of the machine can be ensured in this manner since the machine should move only if this signal is set.

The disturbance range signals must be defined in such a manner that the "1" status always means "Robot outside the disturbance range" (safety against cable breakage).

Within the robot program, this output may be set either via world zones or in the program flow.

If the disturbance range is set in the program flow, collisions must be prevented by timely setting (computer running before the controller).

This can take place either by using a "fine" point or by position-related triggering (`TriggL`, `TriggJ`, `MoveLDO`, or `MoveJDO`) at the end point of the movement.

5 Naming data, I/O, and labels

Prefixes and examples

Type of data	Prefix	Example
aiotrigg	aio	aioExample
bool	b	bPartOK
btnres	res	resExample
busstate	bst	bstExample
buttondata	btn	btnExample
byte	bt	btExample
clock	ck	ckCycleTime
confdata	cf	cfConf15
corrdescr	cd	cdExample
datapos	dp	dpExample
dionum	i	iCondition
dir	dir	dirExample
errdomain	erd	erdExample
errnum	er	erGripperError
errstr	ers	ersExample
errtype	ert	ertExample
event_type	evt	evtExample
extjoint	ej	ejAxpos10
icondata	ico	icoExample
identno	id	idExample
intnum	ir	irCycleStop
iodev	de	deFile
iounit_state	ios	iosExample
jointtarget	jt	jtExample
listitem	lst	lstExample
loaddata	lo	loPart1
loadidnum	lid	lidExample
loadsession	ls	lsExample
mecunit	me	meUnit
motsetdata	mo	moC_Motset
num	n	nCounter
opnum	-	
orient	or	orIent1
paridnum	-	
paridvalidnum	-	

Continues on next page

Continued

Type of data	Prefix	Example
pathrecid	prc	prcExample
pos	ps	psPos1
pose	pe	peFrame1
progdisp	pd	pdSearch
rawbytes	raw	rawExample
restartdata	rsd	rsdExample
robjoint	rj	rjExample
robtarget	p	pHOME
shapedata	sh	shExample
seamdata	sm	smStart1
signalxx	-	Not used in programs
socketdev	sde	sdeExample
socketstatus	sst	sstExample
speeddata	v	v50
stoppointdata	spd	spdExample
string	st	stName
symnum	sy	syStation1
syncident	sid	sidExample
System Data	-	Not defined
taskid	tid	tidExample
tasks	tsk	tskExample
testsignal	-	-
tooldata	t	tGripper
tpnum	tp	tpExample
trapdata	td	tdExample
triggdata	tr	trExample
tunetype	tu	tuExample
Welddata	wd	wdFillet10
weavedata	wv	wvVert
wobjdata	w	wFixture2
wzstationary	wz, (wzs)	wzsExample
wztemporary	wz, (wzt)	wztExample
zonedata	z	z100
Label	lb	lbStart (Labels are not used since the command associated with them, GOTO, should not be used.)

Type of I/O	Prefix	Example
digital in	di	diFetchPart

Continues on next page

Continued

Type of I/O	Prefix	Example
digital group in	gi	giPartNumber
digital out	do	doPartPlaced
digital group out	go	goMoldNumber
analog in	ai	aiTemperature
analog out	ao	aoFeed

This page is intentionally left blank

6 Sample program

6.1 System description

The system consists of an ABB robot IRB 6400/2.8m, one die-casting machine, one parts inspection station, one cooling basin, one trimming press, one chute for good parts and one for rejected parts and scrap.

The robot IRB 6400 removes the molded parts from the die-casting machine and holds them in front of an inspection station. If the part inspected is complete, the die-casting machine is restarted. After inspecting the parts, they are cooled down by turning them over in the cooling basin.

The good (OK) parts are finally placed in a trimming press and removed after trimming in order to place them on the chute for good (OK) parts. The robot then returns to its waiting position in front of the die-casting machine.

If a reject part (scrap) is detected at the inspection station, it is placed after cooling down directly in the chute for rejects (scrap), after which the robot returns to the home position.

If the trimming press is deselected, the parts are placed on the chute for good (OK) parts after they are cooled down.

Special functions

Stop at the end of the cycle:

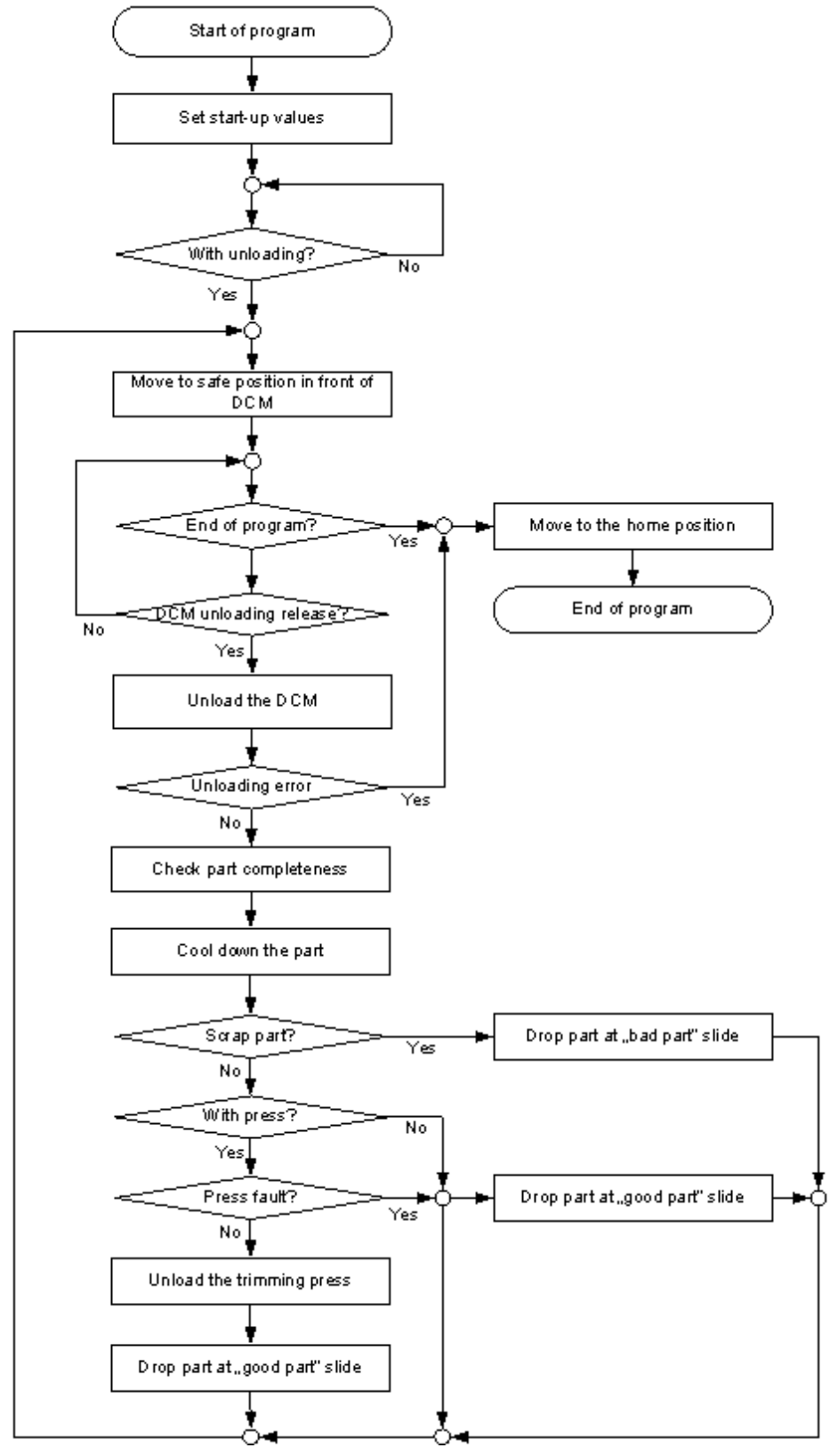
If the operator requests for "Stop at the end of the cycle" using the button on the die-casting machine, the robot removes one part from the die-casting machine, places it down and moves to its home position.

Move the robot to its home position:

If the robot does not get any signals from the peripheral devices, it can be made to cancel the instantaneous processing and move to the home position by pressing the "Robot to home position" button.

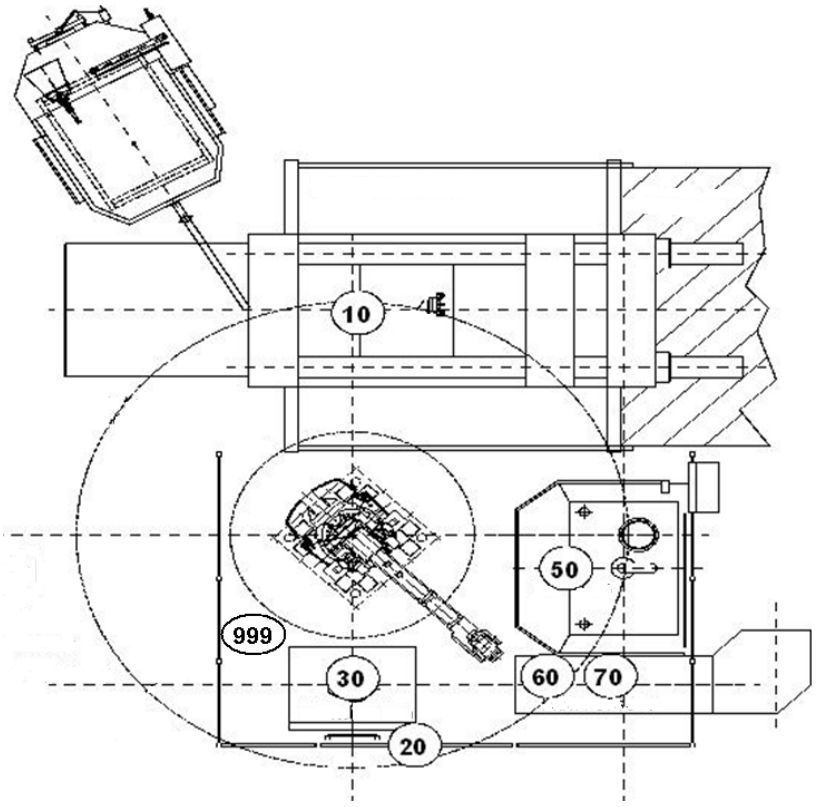
If the robot program is started from `main` and the robot is not in the home position, it can be moved to the home position in manual mode by joystick or on a direct route.

6.2 Flow chart of the system



en1300000182

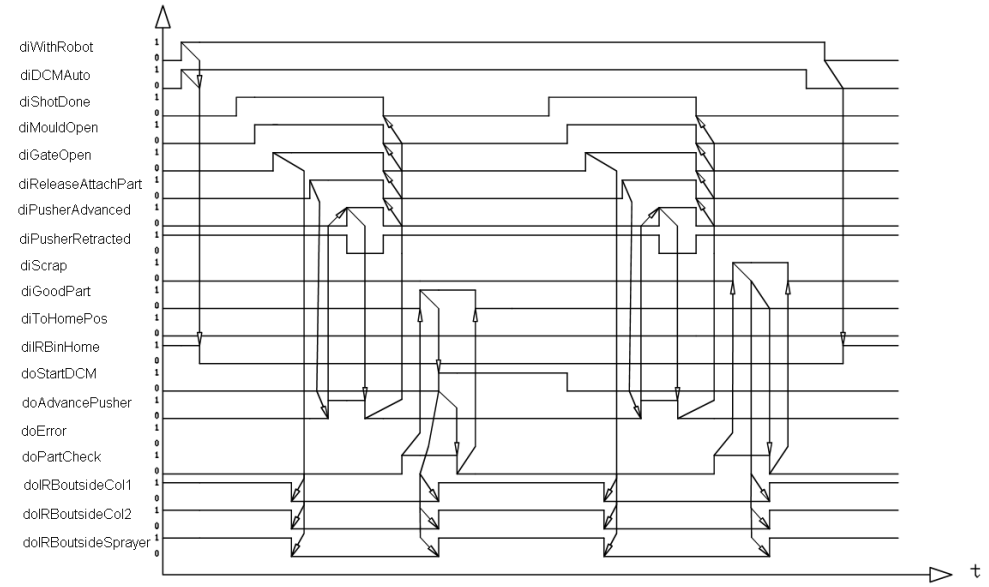
6.3 Overview of the position numbers



en1300000183

Position	Name
10	Preliminary position at the die-casting machine
11	Preliminary position of the mold in the die-casting machine
12	Gripping (picking) position at the die-casting machine
13	Eject position in the die-casting machine
14	End position at the die-casting machine
20	Preliminary position at the inspection station
21	Inspecting position at the inspection station
30	Preliminary position at the cooling basin
31	Cooling position at the cooling basin
50	Preliminary position at the trimming press
51	Placement position at the trimming press
52	Gripping (picking) position at the trimming press
60	Preliminary position at the reject (scrap) chute
61	Placement position at the reject (scrap) chute
70	Preliminary position at the chute for good (OK) parts
71	Placement position at the chute for good (OK) parts
999	Home position

6.4 Signal step diagram



en1300000184

6.5 Signal description (excerpt)

Inputs

Unit	Channel	Signal	Description	SPS
Unit 1	0	diWithRobot	Robot preselected for production	A10.0
Unit 1	1	diDCMAuto	Diecasting machine in automatic mode	A10.1
Unit 1	2	diShotDone	Shot done (Aluminium pushed into the mould)	A10.2
Unit 1	3	diMouldOpen	DCM mould is open	A10.3
Unit 1	4	diGateOpen	DCM robot-sided gate is open	A10.4
Unit 1	5	diReleaseAttachPart	Cores retracted, release to attach part in mould	A10.5
Unit 1	6	diPusherAvanced	Ejector advanced, part pushed out of the mould	A10.6
Unit 1	7	diPusherRetracted	Ejector retracted	A10.7
Unit 1	8	diScrap	Part has been identified as scrap	A11.0
Unit 1	9	diGoodPart	Part has been identified as a good part	A11.1
Unit 1	10	diToHomePos	Request to send robot to home position	A11.2
Unit 1	11	diIRBinHome	Robot is inside the home position	A11.3
Unit 1	...	...	...	...
Unit 2	0	...	...	...
Unit 2	1	...	...	...
Unit 2	2	...	...	...
Unit 2	...	...	...	...

Outputs

Unit	Channel	Signal	Description	SPS
Unit 1	0	doStartDCM	Release to start the next DCM cycle	E10.0
Unit 1	1	doAdvancePusher	Advance pusher to push part out of the mould	E10.1
Unit 1	2	doError	Robot program error	E10.2
Unit 1	3	doPartCheck	Request a part check	E10.3
Unit 1	4	doIRBoutsideCol1	Robot ouside collision area of DCM, channel 1	E10.4
Unit 1	5	doIRBoutsideCol2	Robot ouside collision area of DCM, channel 2	E10.5
Unit 1	6	doIRBouside-Sprayer	Robot outside collision area of the sprayer	E10.6

Continues on next page

6.5 Signal description (excerpt)

Continued

Unit	Channel	Signal	Description	SPS
Unit 1	7	…	…	E10.7
Unit 1	8	…	…	E11.0
Unit 1	9	…	…	E11.1
Unit 1	10	…	…	E11.2
Unit 1	11	…	…	E11.3
Unit 1	…	…	…	
Unit 2	0	…	…	…
Unit 2	1	…	…	…
Unit 2	2	…	…	…
Unit 2	…	…	…	…

6.6 Program printout (excerpt)

```
%%%
VERSION:1
LANGUAGE:ENGLISH
%%%

MODULE DCM1234
!*****************************************************************
!*
!* Cell : DCM extraction cylinder head
!*
!*****************************************************************
!*
!* Customer : ABCD company
!*
!* Project name : V123456
!* Project no. : 47110815
!*
!* Robot no. : IRB 66-12345
!* Software : RobotWare 5.15
!* Revision : 0
!* Key Contr. : ThEgOOdDiEyOuNg
!* Key Drive : AvEcAeSARmoRiTuriTEsAlUTaNT
!*
!* Author : Mr. Bean
!* Company : DCBA Integrators Company
!* San Francisco / California
!* Department : DCBA-EF
!* Telephone : +1 123456789/99 – 0
!*
!* Hotline : +1 123456789/99 – 22
!*
!* Version : 1.0
!* Created : 2013-01-01
!*
!* Modification Date: Name: Reason:
!* 2013-02-31 Mr. Bean New release signal
!*
!*****************************************************************
!
!*********************************************************
!* Boolean declarations
!*********************************************************
!Flag, end the program
VAR bool bProgEnd;
!Flag, part in the parts inspection is not in order
PERS bool bReject:=FALSE;
!*********************************************************
!* System Constants
```

Continues on next page

Continued

```
!*********************************************************
!Constant for the stroke of the ejector at the DGM
CONST num nEjectorStroke:=250;
!Waiting time at the parts inspection
CONST num ntInspection:=2;
!*********************************************************
!* Interrupt Declarations
!*********************************************************
!Interrupt for stop after the end of the cycle
VAR intnum irCycleStop;

!*********************************************************
!* Procedure main
!*
!* Description:
!*
!* This procedure is the main routine and controls the
!* overall execution of the user program.
!*
!* Date: Version: Programmer: Reason:
!* 2012-01-01 1.0 Mr. Bean created
!*********************************************************
PROC main()
!*****************************************
!System data
!Customer: ABCD
!System : DCM Operation
!SN no. : 66-12345
!Author : Mr. Bean, ABB
!Date : 2012-01-01
!Hotline: +1 123456789/99 – 0
!*****************************************
!
!Initialize the start values
Init;
!Check if the gripper is working
GripperTest;
!Move to the home position in front of the DGM
mv99_10;
WHILE NOT bProgEnd DO
Production;
ENDWHILE
mv10_99;
Stop;
ENDPROC

!*********************************************************
!* Procedure Init
!*
!* Description:
!*
```

Continues on next page

Continued

```
!* This procedure sets the initial values for execution
!*
!* Date: Version: Programmer: Reason:
!* 2012-01-01 1.0 Mr. Bean created
!***********************************************************
PROC Init()
!Wait until compressed air is available
WaitDi diAirOK,1;
!Check if the robot is in the home position
CheckHomePos;
!Defined setting of the outputs
Reset doPartGripped;
Reset doDGMUnload;
Reset doEjectBack;
Reset doDGMLoad;
Reset doStartPress;
Reset doBlowOff;
…
…
!Defined setting of the variables
bProgEnd:=FALSE;
!Interrupt for stop after the end of the cycle
IDelete irCycleStop;
CONNECT irCycleStop WITH T_CycleStop;
ISignalDI diCycleStop,1,irCycleStop;
ENDPROC
!***********************************************************
!* Procedure Production
!*
!* Description:
!*
!* This procedure manages the production process.
!*
!* Date: Version: Programmer: Reason:
!* 2012-01-01 1.0 Mr. Bean created
!***********************************************************
PROC Production()
!Unload the die-casting machine
DGMUnload;
mv14_20;
PartCheck;
mv20_30;
CoolDownPart;
!If the parts inspection has reported a reject part
IF bReject THEN
!Dispose of the rejected part
mv30_60;
LoadScrapSlide;
mv60_10;
!Continue working with the trimming press
ELSEIF diWithPress=high THEN
```

Continues on next page

Continued

```
mv30_50;
LoadPress;
UnloadPress;
mv50_70;
LoadGoodPartSlide;
mv70_10;
!Continue working without the trimming press
ELSE
mv30_70;
LoadGoodPartSlide;
mv70_10;
ENDIF
ERROR
IF errno=erCancel RETURN;
ENDPROC

!*********************************************************
!* Procedure LoadScrapSlide
!*
!* Description:
!*
!* Startup parts and parts outside tolerance limits
!* are placed on the slide for rejects (scrap).
!*
!* Date: Version: Programmer: Reason:
!* 2012-01-01 1.0 Mr. Bean created
!*********************************************************
PROC LoadScrapSlide()
!Wait until the slide is ready for loading
WaitDI diScrapSlideFree,high;
!Move to the loading position at the scrap slide
mv60_61;
!Open the gripper
GripperOpen;
!Return to the preliminary position
mv61_60;
bReject:=FALSE;
ENDPROC


!*********************************************************
!* Procedure LoadGoodPartSlide
!*
!* Description:
!*
!* The good parts are placed on the slide for good parts
!*
!* Date: Version: Programmer: Reason:
!* 2012-01-01 1.0 Mr. Bean created
!*********************************************************
PROC LoadGoodPartSlide()
```

Continues on next page

Continued

```
!Wait until the conveyor is ready for loading
WaitDI diIOChuteFree,high;
!Move to the loading position at the conveyor belt
mv70_71;
!Open the gripper
GripperOpen;
!Return to the preliminary position
mv71_70;
ENDPROC


!*********************************************************
!* Procedure T_CycleStop
!*
!* Description:
!*
!* If the input <Stop is active at the end of the cycle>
!* becomes high then a message is output and a flag is set.
!*
!* Date: Version: Programmer: Reason:
!* 2012-01-01 1.0 Mr. Bean created
!*********************************************************
TRAP T_CycleStop
!Input <stop at the end of the cycle> is active
TPWrite stStopCycle;
bProgEnd:=TRUE;
ENDTRAP

ENDMODULE


MODULE Message
!*********************************************************
!*
!* Module name: Message
!*
!*********************************************************
!*
!* Description:
!*
!* This module contains all text messages of the program.
!*
!*
!* Date: Version: Programmer: Reason:
!* 2012-01-01 1.0 Mr. Bean created
!*********************************************************
!
CONST string stStopCycle:= "Stop at the end of the cycle requested";

ENDMODULE
```

Continues on next page

Continued

```
MODULE MOVEMENT

!*********************************************************
!* *
!* Module name: M O V E M E N T *
!* *
!*********************************************************
!* *
!* Description: *
!* *
!* This module contains all movement programs. *
!* *
!* *
!* Date: Version: Programmer: Reason: *
!* 29.02.2007 1.0 Surname,First name created *
!*********************************************************
!
!*********************************************************
!* Tool declarations *
!*********************************************************
PERS tooldata
     tGripper:=[TRUE,[[0,0,0],[1,0,0,0]],[10,[0,0,0],[1,0,0,0],0,0,0]];
!
!*********************************************************
!* Robtarget Definition *
!*********************************************************
CONST robtarget
     p10:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
     p11:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
     p12:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
     p13:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
     p14:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
!
CONST robtarget
     p20:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
     p21:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
!
CONST robtarget
     p30:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
     p31:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
!
CONST robtarget
     p50:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
     p51:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
```

Continues on next page

Continued

```
CONST robtarget
      p52:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
!
CONST robtarget
      p60:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
      p61:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
!
CONST robtarget
      p70:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
CONST robtarget
      p71:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
!
CONST robtarget
      p99:=[[0,0,0],[1,0,0,0],[0,-1,0,0],[9E+009,9E+009,9E+009,9E+009,9E+009,9E+009]];
!
!Position names:
!10 : Preliminary position at the die-casting machine
!11 : Preliminary position of the mold in the die-casting machine
!12 : Gripper position at the die-casting machine
!13 : Eject position at the die-casting machine
!14 : End position at the die-casting machine
!20 : Preliminary position at the inspection station
!21 : Inspecting position at the inspection station
!30 : Preliminary position at the cooling basin
!31 : Cooling position at the cooling basin
!50 : Preliminary position at the trimming press
!51 : Placement position at the trimming press
!52 : Gripper position at the trimming press
!60 : Preliminary position at the chute for rejects (Scrap)
!61 : Placement position at the rejects chute
!70 : Preliminary position at the chute for good (OK) parts
!71 : Placement position at the chute for good (OK) parts
!99 : Home position

PROC mv14_20()
!From : End position at the die-casting machine
!To : Preliminary position at the inspection station
IF OpMode()<>OP_AUTO MoveJ p14,v100,z10,tGripper;
MoveJ p20,v2500,z10,tGripper;
ENDPROC

PROC mv20_30()
!From : Preliminary position at the inspection station
!To : Preliminary position at the cooling basin
IF OpMode()<>OP_AUTO MoveJ p20,v100,z10,tGripper;
MoveJ p30,v2500,z10,tGripper;
ENDPROC

PROC mv30_50()
!From : Preliminary position at the cooling basin
```

Continues on next page

Continued

```
!To : Preliminary position at the trimming press
IF OpMode()<>OP_AUTO MoveJ p30,v100,z10,tGripper;
MoveJ p50,v2500,z10,tGripper;
ENDPROC

PROC mv30_60()
!From : Preliminary position at the cooling basin
!To : Preliminary position at the reject (Scrap) chute
IF OpMode()<>OP_AUTO MoveJ p30,v100,z10,tGripper;
MoveJ p60,v2500,z10,tGripper;
ENDPROC

PROC mv30_70()
!From : Preliminary position at the cooling basin
!To : Preliminary position at the chute for good (OK) parts
IF OpMode()<>OP_AUTO MoveJ p30,v100,z10,tGripper;
MoveJ p70,v2500,z10,tGripper;
ENDPROC

PROC mv50_70()
!From : Preliminary position at the trimming press
!To : Preliminary position at the chute for good (OK) parts
IF OpMode()<>OP_AUTO MoveJ p50,v100,z10,tGripper;
MoveJ p70,v2500,z10,tGripper;
ENDPROC

PROC mv60_10()
!From : Preliminary position at the reject (Scrap) chute
!To : Preliminary position at the die-casting machine
IF OpMode()<>OP_AUTO MoveJ p60,v100,z10,tGripper;
MoveJ p10,v2500,z10,tGripper;
ENDPROC

PROC mv60_61()
!From : Preliminary position at the reject (Scrap) chute
!To : Placement position at the reject (Scrap) chute
IF OpMode()<>OP_AUTO MoveJ p60,v100,z10,tGripper;
MoveJ p61,v2500,z10,tGripper;
ENDPROC
...
...
ENDMODULE
```

Contact us

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

ABB AS, Robotics
Discrete Automation and Motion
Box 265
N-4349 BRYNE, Norway
Telephone: +47 51489000

ABB Engineering (Shanghai) Ltd.
5 Lane 369, ChuangYe Road
KangQiao Town, PuDong District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

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

www.abb.com/robotics

3HAC046417-001, Rev -, en