

ROBOTICS

Application manual

Production Manager



Trace back information:
Workspace 21D version a2
Checked in 2021-11-30
Skribenta version 5.4.005

Application manual
Production Manager

RobotWare 6.13

Document ID: 3HAC052855-001

Revision: C

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, damage 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.

This manual and parts thereof must not be reproduced or copied without ABB's written permission.

Keep for future reference.

Additional copies of this manual may be obtained from ABB.

Original instructions.

© Copyright 2014-2021 ABB. All rights reserved.
Specifications subject to change without notice.

Table of contents

Overview of this manual	7
1 Production Manager	9
1.1 Introduction	9
1.2 Production Manager Execution Engine	10
1.3 Production Manager events	12
1.4 Production Manager Menudata and Partdata	15
2 Production Manager user interface	19
2.1 Overview	19
2.2 Setup menu	22
2.3 Service menu	24
2.3.1 About the Service menu	24
2.3.2 Create a new Setup or Service menu dialog	26
2.3.3 Edit Setup or Service menu	32
2.3.4 Add filtering functionality to menus	34
2.4 Production Information window	35
2.5 Part handler	37
2.5.1 Preview window	39
2.5.2 Create a new part	41
2.5.3 Edit part	47
2.5.4 Test Part	49
2.6 Custom application window	50
2.7 State icons	51
3 Configuring Production Manager	53
3.1 Production Manager Task configuration	53
3.2 Production Manager MultiMove Support	57
3.3 User Authorization System settings	58
4 Production Manager PLC support	61
4.1 How to run Production Manager from PLC	61
4.2 How to run Production Manager from PLC via RAPID	63
5 RAPID references	67
5.1 Instructions	67
5.1.1 ExecEngine - Start execution engine	67
5.1.2 PMgrGetNextPart - Get active part for station in task	68
5.1.3 PMgrSetNextPart - Set active part for station in task	70
5.1.4 PMgrRunMenu - Run menu in task	71
5.2 Functions	72
5.2.1 PMgrAtSafe - Check if task is at safe state	72
5.2.2 PMgrAtService - Check if task is at service state	73
5.2.3 PMgrAtState - Check the state of a task	74
5.2.4 PMgrAtStation - Get the current station for a task	75
5.2.5 PMgrNextStation - Get the next station for a task	76
5.2.6 PMgrTaskNumber - Get the task number	77
5.2.7 PMgrTaskName - Get the task name	78
5.3 Public constants	79
6 Seam Displacement options	81
6.1 General	81
6.2 Starting Seam Displacement option	82
6.3 Functions available in Seam Displacement	84

Table of contents

Index

87

Overview of this manual

About this manual

This manual explains the basics of when and how to use the following *Production Manager* options:

- *Production Manager Execution Engine*
- *Production Manager Events*
- *Production Manager User Interface*
- *Production Manager Menudata and Partdata*
- *Production Manager Configuration*
- *Production Manager MultiMove Support*
- *Production Monitoring*
- *Seam Displacement*

Usage

This manual can be used either as a reference to find out if an option is the right choice for solving a problem, or as a description of how to use an option.

Who should read this manual?

This manual is intended for:

- installation personnel
- robot programmers

Prerequisites

The reader should be familiar with:

- industrial robots and their terminology
- RAPID programming
- system parameters and how to configure them

References

References	Document ID
<i>Safety manual for robot - Manipulator and IRC5 or OmniCore controllerⁱ</i>	3HAC031045-001
<i>Introduction and Safety - Arc Welding Products</i>	<i>Introduction and Safety - Arc Welding Products</i>
<i>Operating manual - IRC5 with FlexPendant</i>	3HAC050941-001
<i>Operating manual - RobotStudio</i>	3HAC032104-001
<i>Operating manual - Getting started, IRC5 and RobotStudio</i>	3HAC027097-001
<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>	3HAC050917-001
<i>Technical reference manual - RAPID Overview</i>	3HAC050947-001
<i>Technical reference manual - System parameters</i>	3HAC050948-001
<i>Application manual - Torch services</i>	3HAC050981-001

Continues on next page

Overview of this manual

Continued

References	Document ID
<i>Application manual - Production Screen</i>	<i>3HAC050964-001</i>

- ⁱ This manual contains all safety instructions from the product manuals for the manipulators and the controllers.

Revisions

Revision	Description
-	Published with RobotWare 6.0.
A	Published with RobotWare 6.05. <ul style="list-style-type: none">• Added section <i>Using signals from EPS on page 56.</i>
B	Published with RobotWare 6.07. <ul style="list-style-type: none">• Updated section <i>Production Manager MultiMove Support on page 57.</i>
C	Published with RobotWare 6.13. <ul style="list-style-type: none">• Updated screenshots from FlexPendant.

1 Production Manager

1.1 Introduction

What is Production Manager?

- Production Manager is process independent middle-layer software running on the IRC5 controller. It is general and it can be used in non-welding applications.
- Production Manager works between the operating system of the robot (RobotWare and related options) and the end user application.
- Production Manager has a highly modular structure that allows customers to plug in applications.

API

Production Manager provides an API for specific applications such as arc welding. Applications running on top of Production Manager can use the production loop together with events and built-in cell logic to facilitate the cell management.

What does Production Manager offer?

Production Manager offers:

- Graphical user interface for running setup and service routines, managing part handling, displaying production information, etc.
- Custom setup and service routines
- Event handling in the production loop
- Station handling
- Rules and definitions on how to develop applications on top of the API
- Part handling

1 Production Manager

1.2 Production Manager Execution Engine

1.2 Production Manager Execution Engine

General

The Production Manager Execution Engine provides an interface that allows external devices to control calls to service and setup routines and calls to user-created part routines.

The Production Manager Execution Engine handles all MultiMove considerations for independent and simultaneous calls (one EE per task).

The Execution Engine will have hooks for user-defined code at critical points in the cycle, such as:

- Before part execution
- After part execution
- Before procedure execution



Tip

See [Production Manager events on page 12](#).

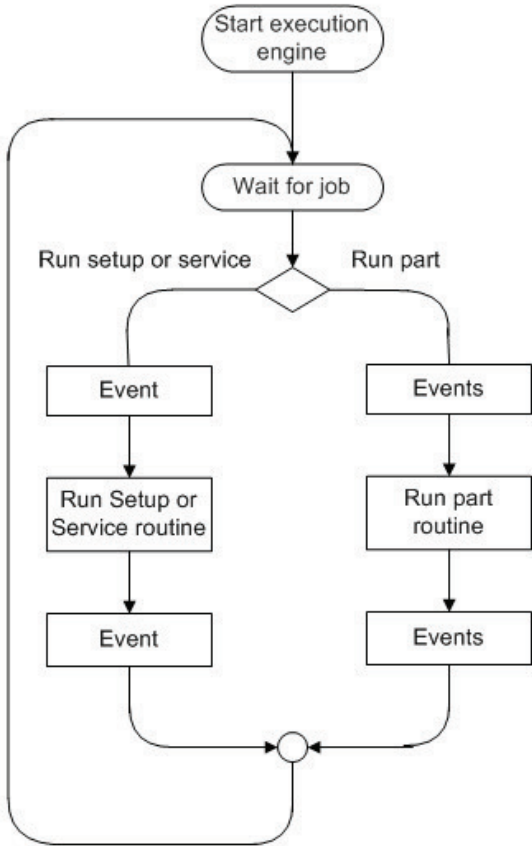
ExecEngine

`ExecEngine` is the *Production Manager Execution Engine* instruction. The instruction takes no arguments and has no-step-in behavior. The user calls this routine from his main routine in each motion task. Typically the user-defined main routine should have a procedure call to `ExecEngine` and nothing else.

The `ExecEngine` routine contains a simple while-loop that monitors the Execution Engine I/O interface for orders specified by the Production Manager GUI or other

Continues on next page

external device such as PLC. The engine is capable of running part routines, setup routines, and service routines.



xx1400002328

1 Production Manager

1.3 Production Manager events

1.3 Production Manager events

General

Production Manager events provide a mechanism for the user to run custom routines at specific points in the `ExecEngine` cycle.

The `ExecEngine` uses event hooks to modularize the application built on Production Manager. There are two ways to hook on to these, a simple and advanced.

Simple Production Manager events

The simple way is to create two procedures named `BeforePart` and `AfterPart` in the tasks that have Production Manager loaded. The procedure `BeforePart` is called just before the part is executed. The procedure `AfterPart` is executed just after the part has been executed.

Advanced Production Manager events

To use the advanced event hooks, data of the type `ee_event` (*execution engine event*) needs to be declared.

```
<dataobject of ee_event>
  <Action of ee_eventnum>
  <ProcName of string>
  <taskList of string>
  <SortOrder of byte>
  <validStation of byte>
```

List of ee_event

ee_event	Description
ee_eventnum action	The type of event to subscribe to. See list of available events in List of available events on page 12 .
string procName	The actual procedure that will be called when running this event.
string taskList	The task names separated by ':' for the tasks where this routine will be executed. If more than one task is entered it means that these tasks will be executed simultaneously. Example: T_ROB1:T_ROB2.
byte sortOrder	Value from 0-255 defining in which order this event will be executed with respect to other events of the same action type.
byte validStation	Value from 0-255 defining for which station this event is valid.

List of available events

Event	Value	Description
EE_START	1	EE_START is called when <code>ExecEngine</code> is called. NOTE! This event does not have a valid station.
EE_CYCLE_START	2	EE_CYCLE_START is called when Production Manager receives an order to execute a part. NOTE! The <code>validStation</code> element works on <code>AtStation</code> , not <code>NextStation</code> .

Continues on next page

1 Production Manager

1.3 Production Manager events

Continued

Event	Value	Description
EE_PROC_START	3	EE_PROC_START is called before a setup or service routine is executed. NOTE! The validStation element works on AtStation, not NextStation.
EE_PRE_PROD	4	Called before part is executed
EE_CLOSE_JIG	5	Called before part is executed
EE_INDEX	6	Called before part is executed
EE_PRE_PART	7	Called before part is executed
(part execution)		
EE_POST_PART	8	Called after part is executed
EE_OPEN_JIG	9	Called after part is executed
EE_SERVICE	10	Called after part is executed
EE_POST_PROD	11	Called after part is executed
EE_ABORT	12	This event is launched if the production is aborted due to an error. NOTE! This event does not have a valid station.
EE_WAIT_ORDER	13	EE_WAIT_ORDER is called repeatedly when Production Manager is waiting for an order. NOTE! This event does not have a valid station.
EE_POST_PROC	14	EE_POST_PROC is called after a setup or service routine is executed. NOTE! The validStation element works on AtStation, not NextStation.

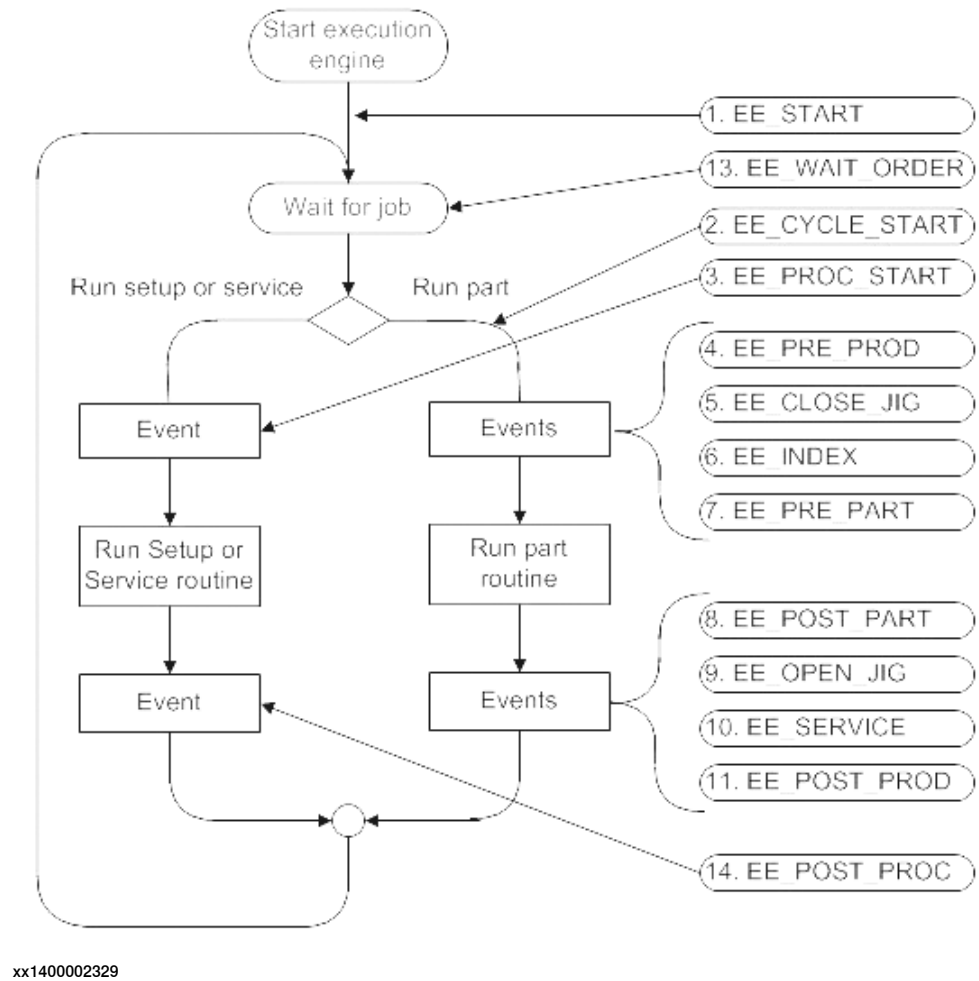
Continues on next page

1 Production Manager

1.3 Production Manager events

Continued

Event order



1.4 Production Manager Menudata and Partdata

Menudata

Menudata	Description
string description	A short description of the menu item.
string image	The name of the icon to use next to the menu item.
string procName	The actual procedure that will be called when running this setup or service routine. The <code>procName</code> can also point to a procedure in a module that resides on the controller's file system, see Dynamic parts and menus on page 16 .
byte validStation	Value from 0-255 defining to which station this menu item will be visible.
string taskList	The task names separated by ':' for the tasks where this routine will be executed. If more than one task is entered it means that these tasks will be executed simultaneous. Example: T_ROB1:T_ROB2.
byte validPosition	Value from 0-255 defining at which robot position this menu item will be visible. Three predefined positions are available: <ul style="list-style-type: none"> GAP_SHOW_SAFE GAP_SHOW_SERVICE GAP_SHOW_ALWAYS
bool allowAfterError	True if the menu item should be shown after an error.
num type	1 = Setup menu, 2 = Service menu.
byte minUserLevel	Value from 0-255 defining the minimum user level required to run this menu, if using UAS. See User Authorization System settings on page 58 for more information.
bool blockOtherTasks	If this is set to <code>True</code> all other tasks will be blocked during the execution of this routine.
num plcCode	Unique identifier index for PLC interfaces. Note: Used only when an external system is controlling the robot system.

Partdata

Partdata	Description
string pathProcName	The procedure that will be called when running this part. The <code>pathProcName</code> can also point to a procedure in a module that resides on the controller's file system, see Dynamic parts and menus on page 16 .
string description	A short description of the part.
string taskList	The task names separated by ':' for the tasks where this part will be executed. If more than one task is entered it means that these tasks will be executed simultaneous. Example: T_ROB1:T_ROB2.
byte validStation	Value from 0-255 defining for which station this part will be valid.

Continues on next page

1 Production Manager

1.4 Production Manager Menudata and Partdata

Continued

Partdata	Description
num plcCode	Unique identifier index for PLC interfaces. Note: Used only when an external system is controlling the robot system
string image	The name of the picture to use in the preview frame when browsing parts in the Part handler.
string advPart	This represents a reference to an advanced part. See Example 1, advanced part on page 16 .

Example 1, advanced part

The example below shows how the advanced part field in the `partdata` can be used.

The `advPart` field works as a reference to a custom data object. The referenced data object often represents application specific data.

This is the `partdata` instance with the reference to an advanced part called `pdvProgStn1`.

```
TASK PERS partdata pdProgStn1:= ["ProgStn1","Program station 1",  
    "T_ROB1:T_ROB2:T_ROB3:T_POS1",1, "MyPart.gif","pdvProgStn1"];
```

The `partadv` is a custom data type in RAPID. In this example the application takes advantage of process angles, load angles and load data.

```
RECORD partadv  
    extjoint procAngle;  
    extjoint loadAngle;  
    loaddata Load;  
ENDRECORD
```

Below is the declaration of the `pdvProgStn1` data instance.

```
TASK PERS partadv pdvProgStn1:=  
    [[0,0,0,0,0,0],[0,0,0,0,0,0],[0,[0,0,0],[0,0,0,0],0,0,0]];
```

The application running on top of Production Manager can process the `partdata` and take advantage of the advanced part data, for example, to move the external axis to a certain process and load angle before Production Manager is ordered to produce the part.

Dynamic parts and menus

A menu or a part can be loaded dynamically by using a file path in the menu's `procName` or the part's `pathProcName` field. This feature is useful for saving memory since the module will be loaded, executed and unloaded in runtime. All changes in the loaded module will be saved when it is unloaded. This dynamic feature works for both PLC orders and normal operator initiated actions.

Continues on next page

The path to the module and the procedure are delimited with a '@'. Below is an example of a dynamic part:

```
TASK PERS partdata pdDynProgStn1 :=  
    ["HOME:/DynPart/DynPartPrcR1S1.mod@PartStn1", "Dynamic part  
sta- tion 1", "", 1, 0, "", ""];
```

```
%%%
```

```
VERSION: 1
```

```
LANGUAGE: ENGLISH
```

```
%%%
```

```
MODULE DynPartPrcR1S1
```

```
PROC PartStn1()
```

```
<SMT>
```

```
ENDPROC
```

```
ENDMODULE
```

A dynamic menu could look like this:

```
CONST menudata mdDynMenu := ["Run dynamic  
menu", "", "HOME:/DynPart/DynMenuPrc.mod@DynMenuProc", 255, "",  
GAP_SHOW_ALWAYS, TRUE, GAP_SERVICE_TYPE, 0, FALSE, 0];
```

```
%%%
```

```
VERSION: 1
```

```
LANGUAGE: ENGLISH
```

```
%%%
```

```
MODULE DynMenuPrc
```

```
PROC DynMenuProc()
```

```
<SMT>
```

```
ENDPROC
```

```
ENDMODULE
```

This page is intentionally left blank

2 Production Manager user interface

2.1 Overview

The Production Manager user interface requires some UAS grants to operate properly. See [User Authorization System settings on page 58](#) for detailed information.

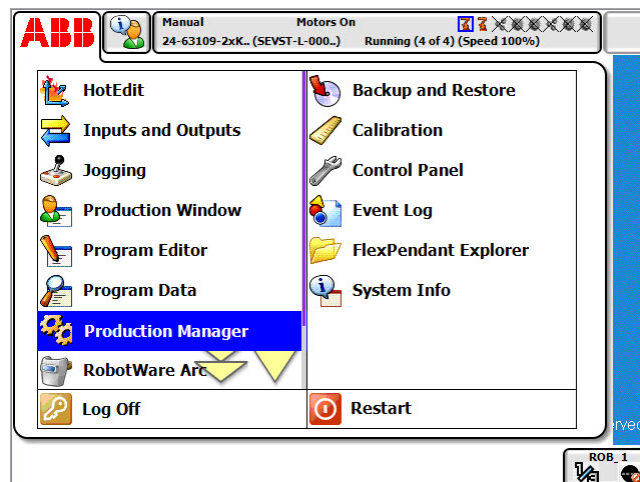
Some of the available features, like menus and parts, utilize an icon or image attribute to increase the usability. Graphical resources can be added to the FlexPendant by following the steps below.

Image deployment steps:

- 1 Open a FTP client session with the controller.
- 2 Navigate to the system you want to update images.
- 3 Copy the graphical resources into the system directory.

Accessing different functions

- 1 Go to the **ABB** menu and launch the Production Manager.



xx1400002330

Continues on next page

2 Production Manager user interface

2.1 Overview

Continued

Starting view:

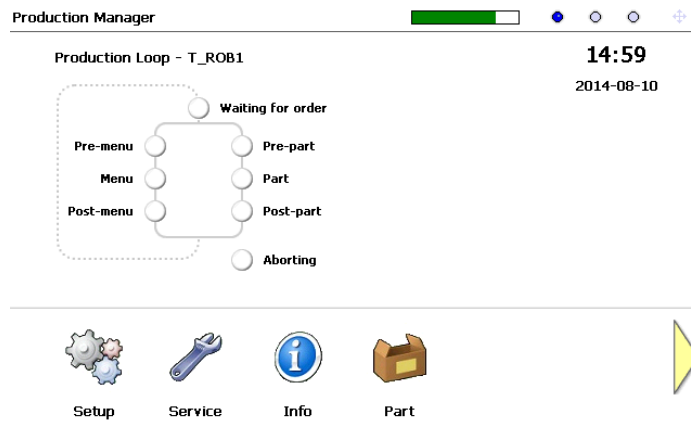


xx1400002331

In the Production Manager main menu all production related functions can be accessed.

Production Screen

Production Manager can also be run within the Production Screen framework. The GUI functionality is added automatically to the Production Screen desktop, as apps, when both options are installed. The desktop can then be extended with custom apps and widgets using the flexibility of the customizable interface in Production Screen.



xx1400002402



Note

Production Screen requires a separate option.

For more information, see *Application manual - Production Screen*.

Continues on next page

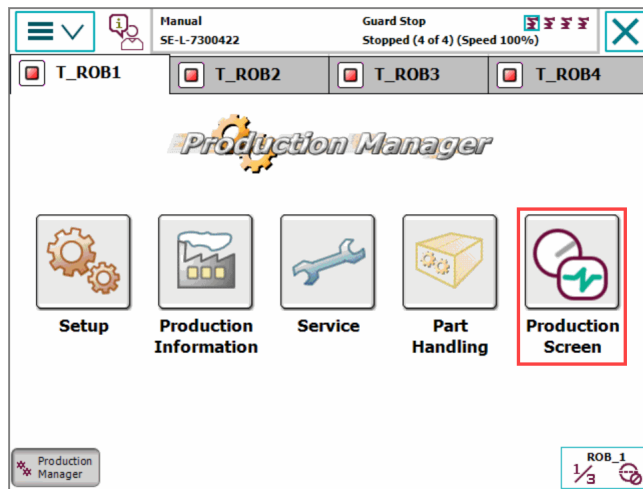
MultiMove system

In a MultiMove system the different tasks and robot are displayed as tabs at the top of the page.

In this example the cell is loaded with three robots and a positioner.

- 1 In the Production Manager main menu:

To explore the functions for the positioner, simply select the IRBP 1 tab.



xx1400002332

2 Production Manager user interface

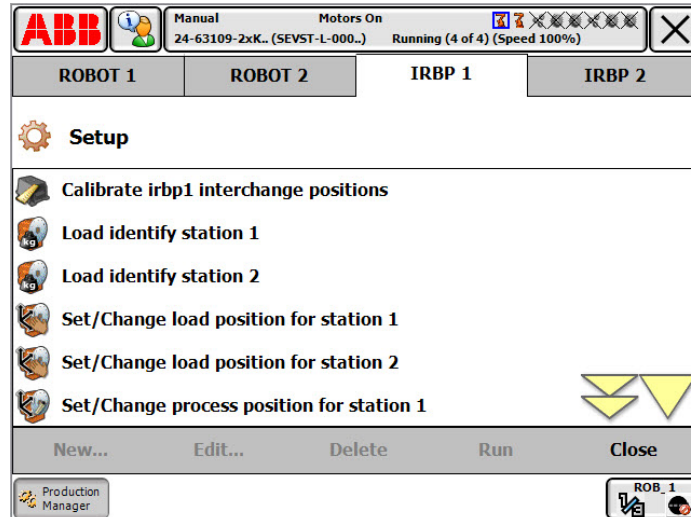
2.2 Setup menu

2.2 Setup menu

Overview

The **Setup** menu contains functions for setting up the robot, positioner or cell. The following figure shows the Setup procedures available for the positioner.

The functions available in **Setup** may be restricted by the User Authorization System, see [User Authorization System settings on page 58](#).



xx1400002333

Add

It is possible to add custom setup procedures in the list. This is done by adding **RAPID** variables of type `menudata` to the appropriate task. All variables of type `menudata`, and declared with `menudata.type = GAP_SETUP_TYPE`, will automatically be added to the list in the **Setup** window. See [Production Manager Menudata and Partdata on page 15](#) for details about the `menudata` type.

It is also possible to add Setup menus from the user interface by tapping **New...** on the command bar. See [Create a new Setup or Service menu dialog on page 26](#).

Edit

To edit the selected menu item, tap **Edit...** on the command bar. See [Edit Setup or Service menu on page 32](#).

Delete

To delete the selected menu item, tap **Delete** on the command bar.

Launch

To launch a Setup procedure simply select the line and tap **Run** on the command bar. See [Edit Setup or Service menu on page 32](#).

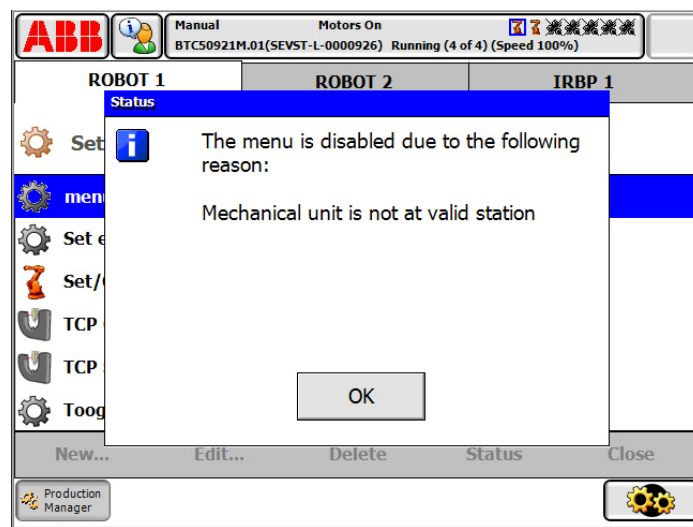
Continues on next page

Enable/disable

The setup procedures can be disabled depending of the state of the task. A Setup item is enabled if:

- the execution state must be in running mode.
- the `menudata` fields `validStation`, `validPosition`, `allowAfterError` and `minUserLevel` must all be valid for an item to be enabled. The value of these fields depends on the state of the cell and the user's grant level. Production Manager must also be in running and ready mode in order for the Setup menus to be enabled.

If a menu is disabled the command bar item **Run** will change to **Status**. If **Status** is tapped, a message box will appear explaining why the menu is disabled.



xx1400002334

2 Production Manager user interface

2.3.1 About the Service menu

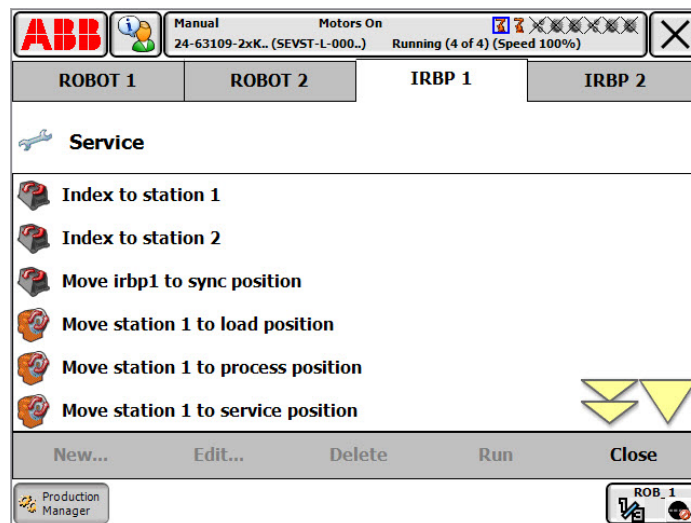
2.3 Service menu

2.3.1 About the Service menu

Overview

The **Service** menu contains functions for running service procedures for the robot, positioner or cell. The following figure shows the service procedures available for the positioner.

The functions available in **Service** may be restricted by the User Authorization System, see [User Authorization System settings on page 58](#).



xx1400002335

Add

It is possible to add custom service procedures in the list. This is done by adding **RAPID** variables of type `menudata` to the appropriate task. All variables of type `menudata`, and declared with `menudata.type = GAP_SERVICE_TYPE`, will automatically be added to the list in the **Setup** window. See [Production Manager Menudata and Partdata on page 15](#) for details about the `menudata` type.

It is also possible to add Service menus from the user interface by tapping **New...** on the command bar. See [Create a new Setup or Service menu dialog on page 26](#).

Edit

To edit the selected menu item, tap **Edit...** on the command bar. See [Edit Setup or Service menu on page 32](#).

Delete

To delete the selected menu item, tap **Delete** on the command bar.

Launch

To launch a service procedure simply select the line and tap **Go** on the command bar.

Continues on next page

Enable/disable

The service procedures can be disabled depending of the state of the task. A service item is enabled if:

- Execution state must be in running mode.
- The `menudata` fields `validStation`, `validPosition`, `allowAfterError` and `minUserLevel` must all be valid for an item to be enabled. The value of these fields depends on the state of the cell and the user's grant level. Production Manager must also be in running and ready mode in order for the Service menus to be enabled.
- If a menu is disabled the command bar item **Run** will change to **Status**. If **Status** is tapped, a message box will appear explaining why the menu is disabled.

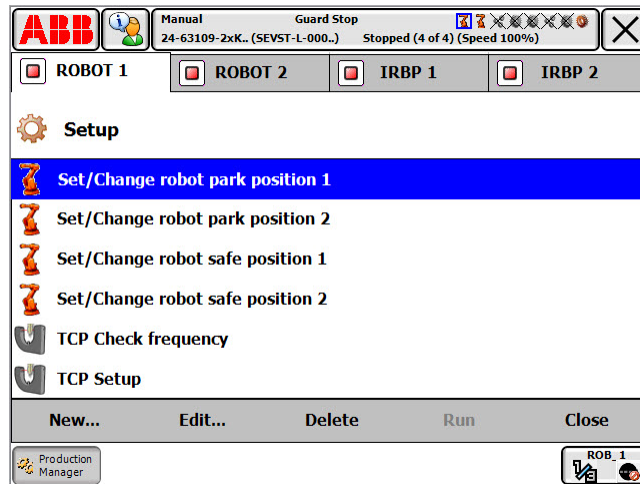
2 Production Manager user interface

2.3.2 Create a new Setup or Service menu dialog

2.3.2 Create a new Setup or Service menu dialog

Procedure

- 1 In the Production Manager main menu select **Setup** or **Service** depending on which type of menudata you would like to create.
- 2 Tap **New** on the command bar.

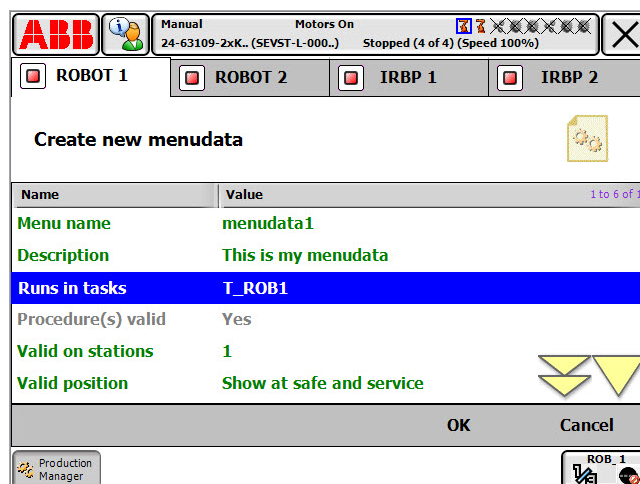


xx1400002336

- 3 The **Create new menudata** dialog has a number of fields to enter, see [New menudata dialog on page 27](#). The user interface will help the user to create a new menudata instance together with the actual procedure to be called when the menu is executed.

The menudata instance will be declared as `TASK PERS.` The procedure will be an empty `PROC` ready to be added with instructions.

Some of the fields in the dialog are loaded with default values.



xx1400002337

Continues on next page

2 Production Manager user interface

2.3.2 Create a new Setup or Service menu dialog

Continued

New menudata dialog

Default value	Description
Procedure name	This is the procedure that will be called when menu is executed. A default name is suggested when creating new <code>menudata</code> . Select the field to change the suggested name by typing a new name in the alpha pad.
Description	A custom string that describes the Setup or Service menu. This is the name that will be displayed in the Setup or Service window list.
Run in tasks	These are the tasks this setup or service procedure should be declared in. Select the line and check the boxes in the window that appears on the right side. If more than one task is simultaneously selected it means that these tasks will be executed simultaneous.
Procedure valid	This field cannot be edited. It indicates if the procedure name and menudata instance name is valid in all tasks in the task list.
Valid on stations	Select the stations this setup or service procedure will be valid on.
Valid position	Select the position this setup or service procedure will be valid for. Three predefined positions are available.
Block other tasks	If this is set to True all other tasks will be blocked during the execution of this routine.
PLC code	Unique identifier index for PLC interfaces.
User Level	Value from 0-255 defining the minimum user level required to run this menu, if using UAS. See User Authorization System settings on page 58 for more information.
Menudata instance	The name of the menudata instance in RAPID.
Declared in module	Select the module where the data and the procedure will be declared. It is possible to create a new module for the menu. If a task list is used, the module will be created in all tasks in the task list if it does not already exist, and the <code>menudata</code> and procedure will be placed in this module. Note: Only normal program modules will be visible in the list.

Create a new dynamic Setup or Service menu dialog

- 1 In the Production Manager main menu select **Setup** or **Service** depending on which type of menudata you would like to create.

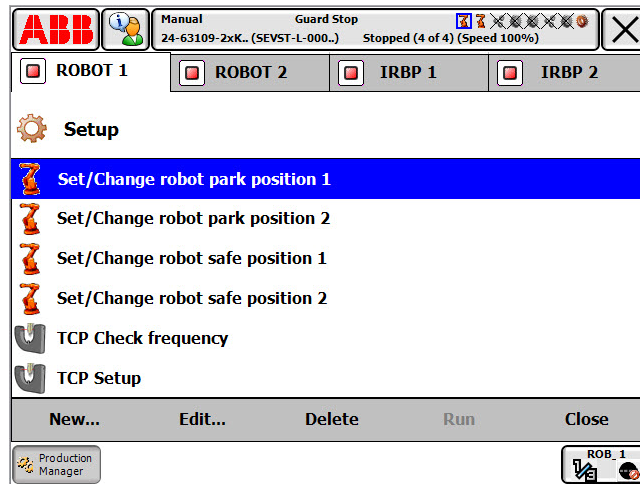
Continues on next page

2 Production Manager user interface

2.3.2 Create a new Setup or Service menu dialog

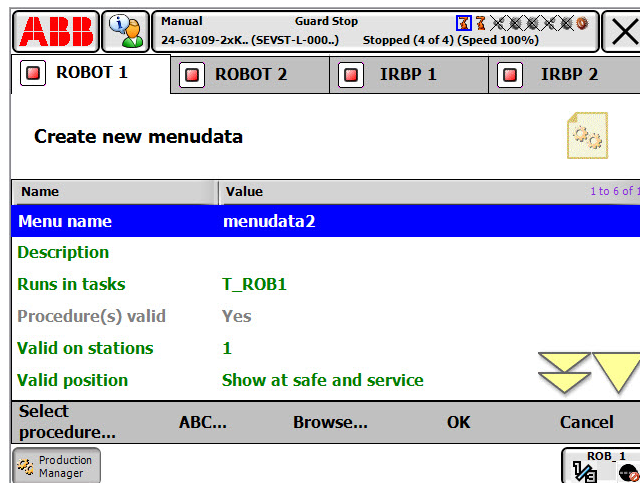
Continued

2 Tap New on the command bar.



xx1400002336

3 Select Menu name in list and tap ABC....



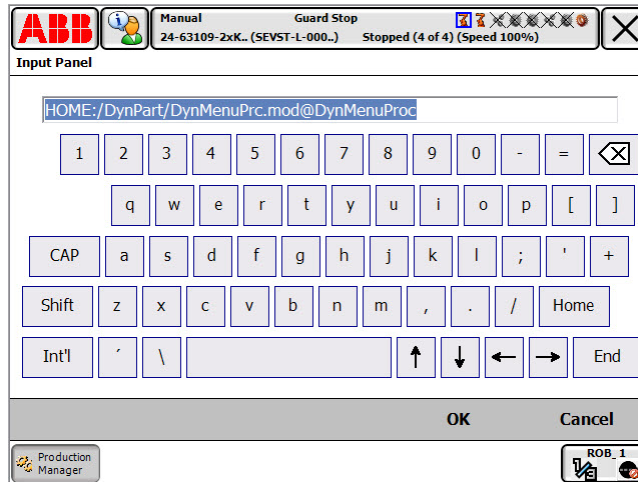
xx1400002338

Continues on next page

2 Production Manager user interface

2.3.2 Create a new Setup or Service menu dialog Continued

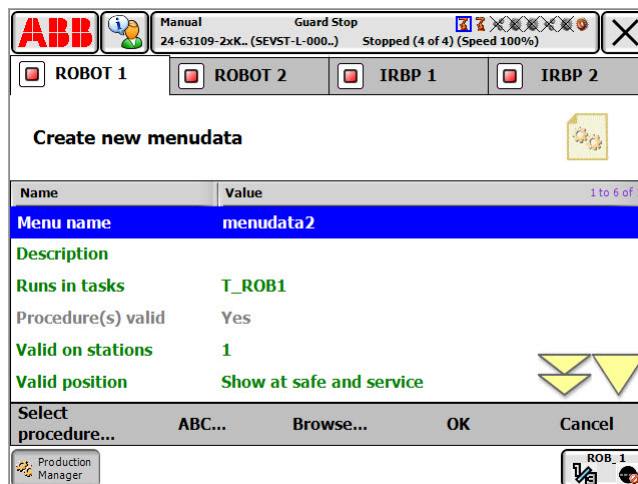
- 4 Enter the path to the module and procedure name separated with @. If module does not exist, it will be created.



xx1400002339

- 5 Or

Select Menu name in list and tap **Browse...**



xx1400002340

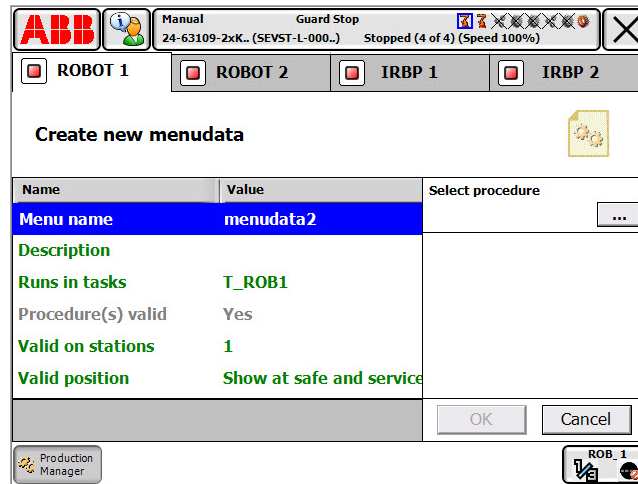
Continues on next page

2 Production Manager user interface

2.3.2 Create a new Setup or Service menu dialog

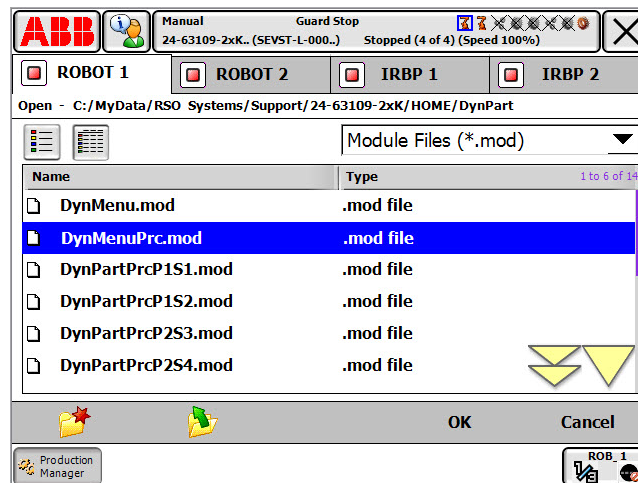
Continued

6 Tap button ... to browse for module.



xx1400002341

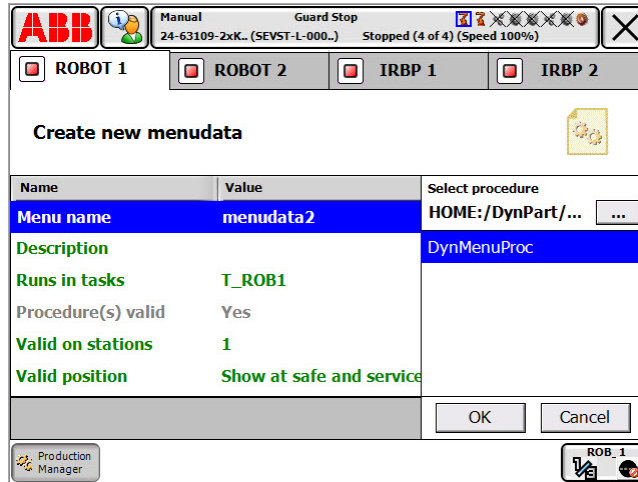
7 Select module.



xx1400002342

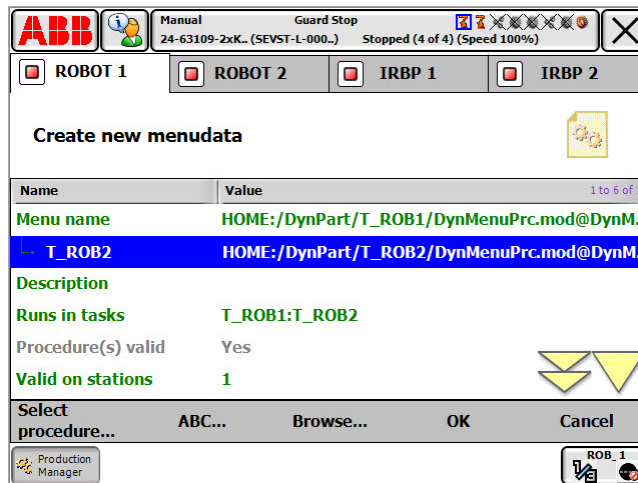
Continues on next page

8 Select procedure and tap OK.



xx1400002343

9 If menu is synchronized in several tasks, select task in list and repeat step 4 or step 5 to 8.



xx1400002344

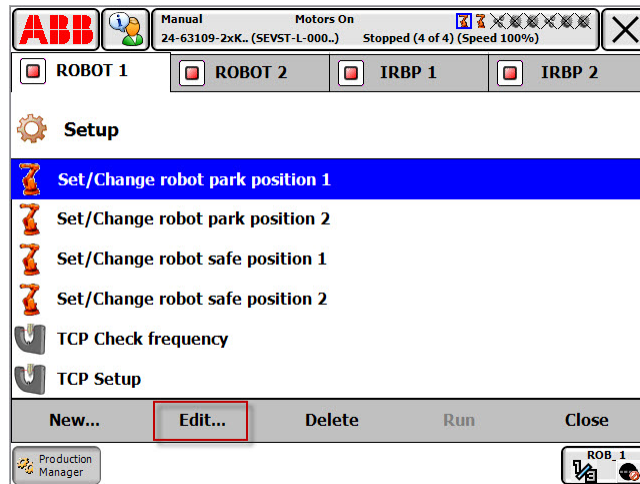
2 Production Manager user interface

2.3.3 Edit Setup or Service menu

2.3.3 Edit Setup or Service menu

Procedure

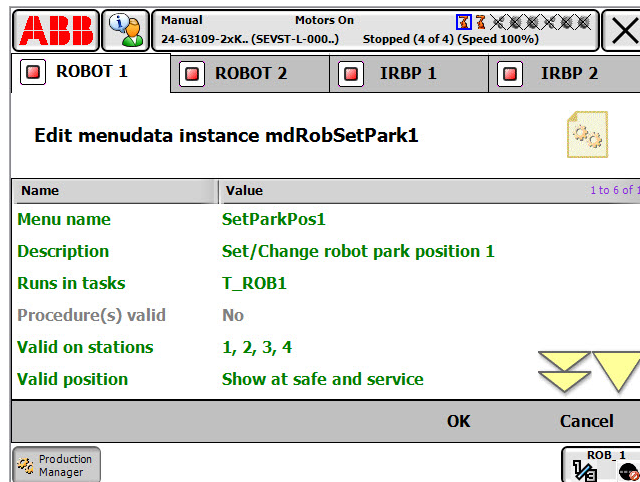
- 1 In the Production Manager main menu select **Setup** or **Service** depending on which type of menudata you would like to edit.
- 2 Select the menu to edit.



xx1400002345

- 3 Tap **Edit**.

The **Edit** menudata dialog has a number of fields to enter. See [Edit Setup or Service menu on page 32](#). The fields will be loaded with data for the current menudata.



xx1400002346

Continues on next page

Edit Setup or Service menu dialog

Default value	Description
Procedure name	This is the procedure that will be called when menu is executed. It is not possible to specify a new menudata name, only select from an already existing menu. Use the dropdown list to search for procedures in another module..
Description	A custom string that describes the Setup or Service menu. This is the name that will be displayed in the Setup or Service window list.
Run in tasks	These are the tasks this setup or service procedure should be declared in.
Select the line and check the boxes in the window that appears on the right side.	If more than one task is simultaneously selected it means that these tasks will be executed simultaneous.
Procedure valid	This field cannot be edited. It indicates if the procedure name and menudata instance name is valid in all tasks in the task list.
Valid on stations	Select the stations this setup or service procedure will be valid on.
Valid position	Select the position this setup or service procedure will be valid for. Three predefined positions are available.
Block other tasks	If this is set to True all other tasks will be blocked during the execution of this routine.
PLC code	Unique identifier index for PLC interfaces.
User Level	Value from 0-255 defining the minimum user level required to run this menu, if using UAS. See User Authorization System settings on page 58 for more information.
Menudata instance	The name of the <code>menudata</code> instance in RAPID. This field is not possible to edit.
Declared in module	The module where the data and the procedure will be declared. Note: This field is not possible to edit.

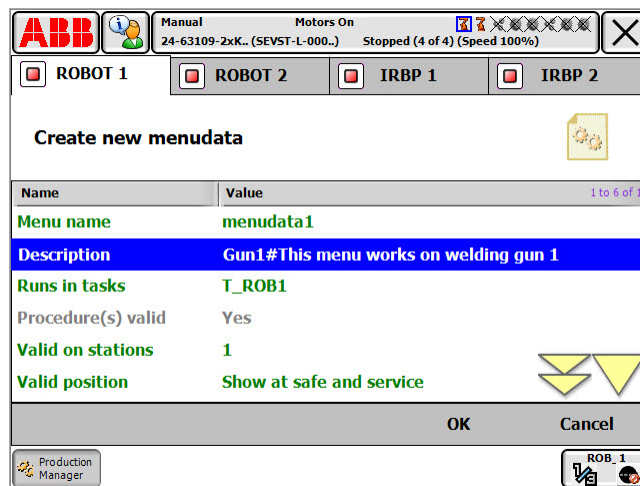
2 Production Manager user interface

2.3.4 Add filtering functionality to menus

2.3.4 Add filtering functionality to menus

It is possible to tag Setup and Service menus with a category and apply a filter on these categories. This one-level filtering is achieved by introducing a special syntax in the description field of the `menudata` instance.

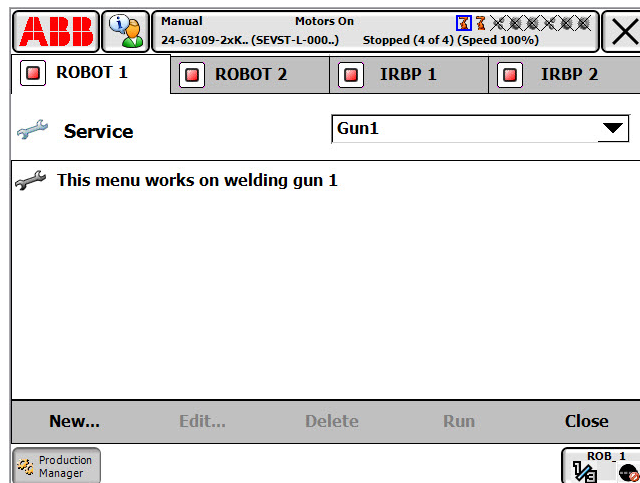
- 1 In the Production Manager main menu select **Setup** or **Service** depending on which type of `menudata` you would like to add or edit.
- 2 Select the menu to edit or tap **New...** to create a new menu.
- 3 In the **Description** field, add the following syntax to add a filter: "categoryname"#"description string".



Name	Value
Menu name	menudata1
Description	Gun1#This menu works on welding gun 1
Runs in tasks	T_ROB1
Procedure(s) valid	Yes
Valid on stations	1
Valid position	Show at safe and service

xx1400002347

- 4 Use the drop-down box to filter the menus based on the categories.



Service: Gun1

This menu works on welding gun 1

New... Edit... Delete Run Close

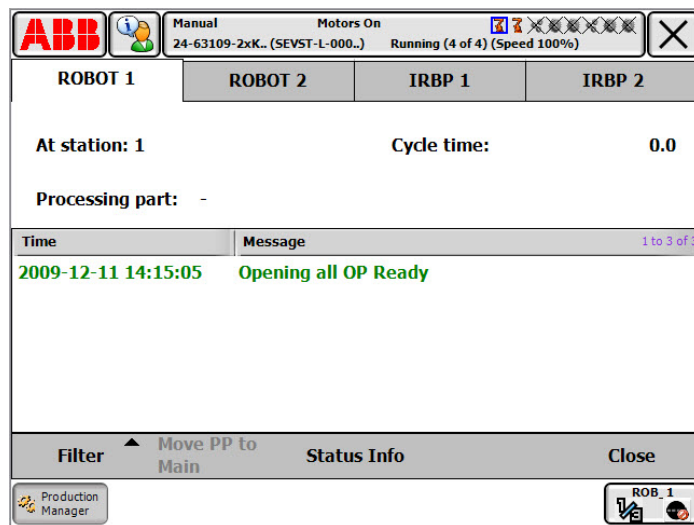
xx1400002348

2.4 Production Information window

Overview

The production information window displays information during the production of parts.

- Current station and active part are displayed.
- Messages sent from the active task are displayed in the lower part of the window.
- By tapping **Move PP to Main**, it is possible to move the program pointer to main in all tasks. A log of the 50 latest messages is saved and can be viewed at any time.



xx1400002349



Note

It is possible to see messages sent from other tasks by checking the corresponding taskname in the **Filter** menu.

Status information

The status information window displays detailed information during the production cycle.

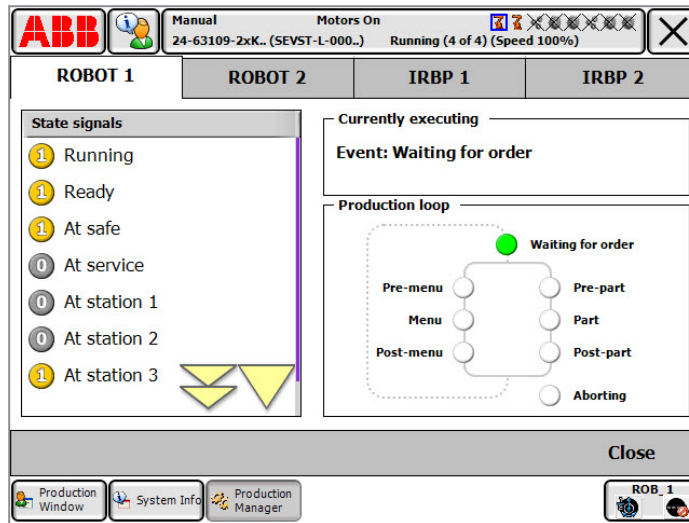
- State of configured signals.
- What event executing.
- What procedure executing. If no procedure given and part has `tasklist` defined, the task is waiting/synchronizing for other task in list.
- Graphical view of production cycle

Continues on next page

2 Production Manager user interface

2.4 Production Information window

Continued



xx1400002350

2.5 Part handler

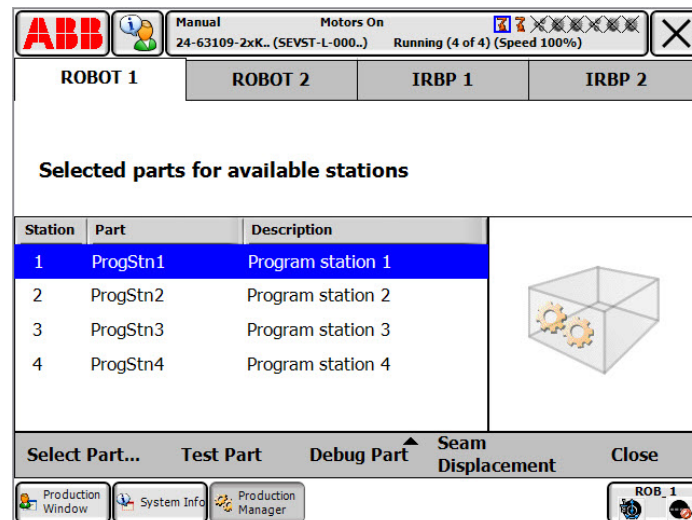
Overview

The functions available in Part Handler windows may be restricted by the User Authorization System, see [User Authorization System settings on page 58](#).

Select parts for available stations

The part handler window lets the user select parts for available stations

- 1 Select the station number.
- 2 Tap **Select Part** to select a new part to this station.



xx1400002351

Debug part

When a part has been selected for a station it is possible to run the part in debug mode. When running a part in debug mode the execution will stop just before the user code is called at every place in the production loop. Typically the user presses **Step forward** when the execution has stopped. This feature can be useful if an error has occurred during the process and it is too expensive to throw away the object. The operator can fix the problem, step through the program until he reaches the error point and then continue production.

Two different debug types are available:

- **Debug all**
The execution will stop before all user code calls throughout the complete part cycle.
- **Debug to part**
The execution will stop before all user code calls for the events before the part (part procedure call included). When the part procedure has been invoked the user can press Start to continue normal execution without any stops at the events after the part.

Continues on next page

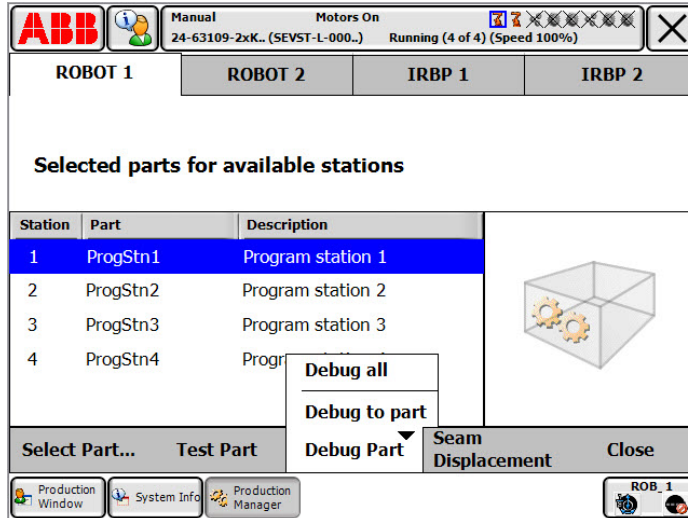
2 Production Manager user interface

2.5 Part handler

Continued

To start debugging:

- 1 Select the station number.
- 2 Tap **Debug Part** on the command bar and make your selection to start debugging.

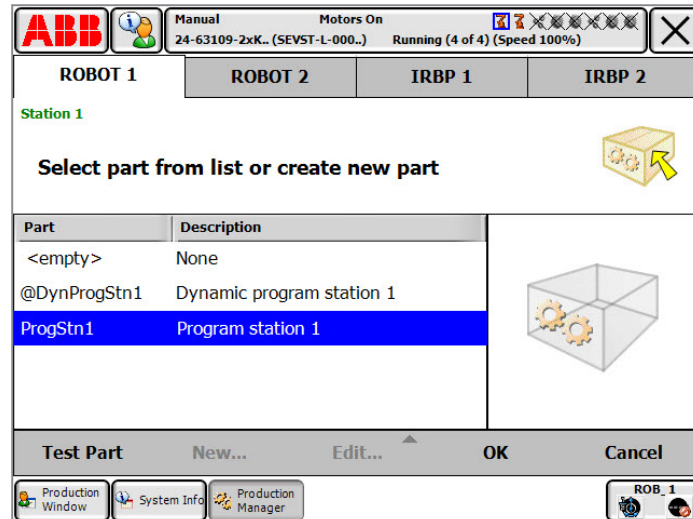


xx1400002352

Continues on next page

2.5.1 Preview window

If the image field is entered in the `partdata`, the preview window on the right will display a picture of the part when a station is selected.



xx1400002353

Deploying image resources to the FlexPendant:

- 1 Open a FTP client session with the controller.
- 2 Navigate to the system you want to update images.
- 3 Copy the graphical resources into the system directory.

Select

To select a part, select the line and tap **OK**.

If the part contains a task list, this part will be selected for this station in all tasks in the task list, if the `partdata` instance exists in the other tasks. If the part procedure name cannot be found in a task, a message box will be displayed where it is possible to select the part anyway and override the warning.



Note

Parts with tasklists are connected via the `partdata` instance name, not the part procedure name. I.e. it is possible to have different names on the part procedures between synchronized parts as long as the `partdata` instance name is the same. Only persistent `partdata` instances will be shown in the list of available parts.

Create

To create a new part, tap **New...** on the command bar. See [Create a new part on page 41](#).

Edit

To edit the selected part, tap **Edit...** on the command bar. See [Part Handler - Edit part on page 47](#).

Continues on next page

2 Production Manager user interface

2.5.1 Preview window

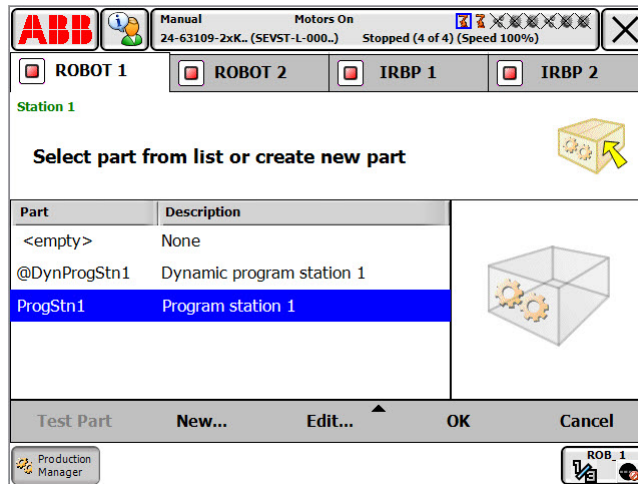
Continued

Delete

To delete the selected part, tap **Delete** on the command bar.

2.5.2 Create a new part

- 1 In the Production Manager main menu select **Part handling**.
- 2 Tap **New**.

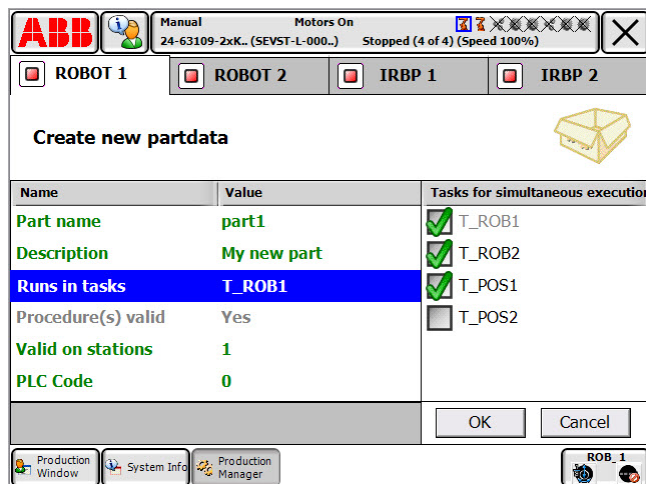


xx1400002354

- 3 The **Create new partdata** dialog has a number of fields to enter, see [New parts dialog on page 42](#). The user interface will help the user to create a new partdata instance together with the actual part procedure to be called during production.

The partdata instance will be declared as `TASK PERS`. The part procedure will be an empty `PROC` ready to be added with instructions.

Some of the fields in the dialog are loaded with default values.



xx1400002355

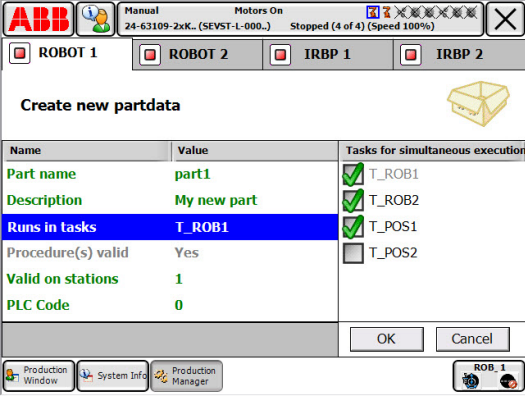
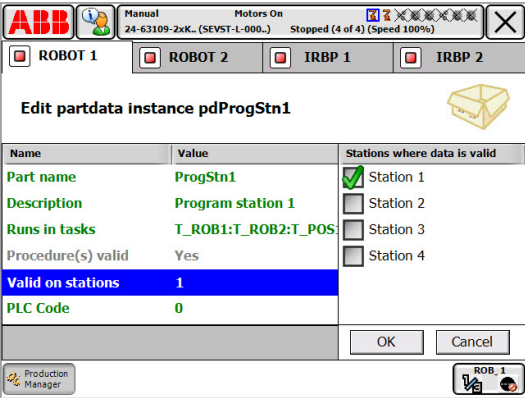
Continues on next page

2 Production Manager user interface

2.5.2 Create a new part

Continued

New parts dialog

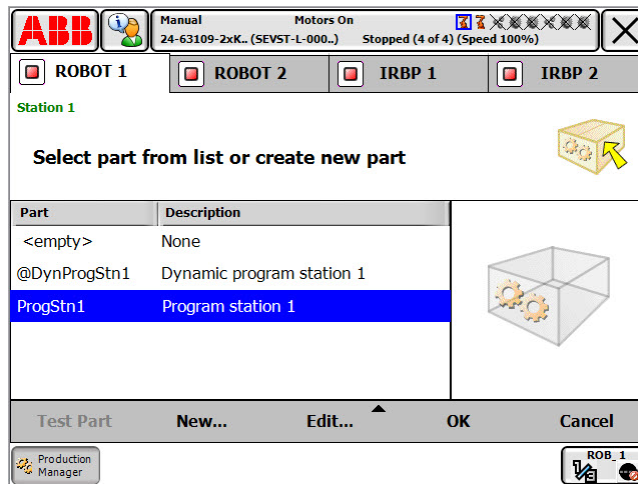
Default value	Description
Part name	<p>This is the procedure that will be called during production. It typically contains process instructions such as <code>ArcL</code>, <code>SpotL</code>. A default name is suggested when creating new <code>partdata</code>.</p> <p>Select the field to change the suggested name by typing a new name in the alpha pad.</p>
Description	<p>A custom string that describes the part.</p>
Run in tasks	<p>These are the tasks this part should be declared in.</p> <p>Select the line and check the boxes in the window that appears on the right side.</p> <p>If more than one task is selected it means that these tasks will be executed simultaneously.</p>  <p>xx1400002355</p>
Procedure valid	<p>This field cannot be edited. It indicates if the Part name and <code>partdata</code> instance name are valid in all tasks in the task list.</p>
Valid on stations	<p>Select the stations this part will be valid on.</p>  <p>xx1400002356</p>
Partdata instance	<p>The name of the <code>partdata</code> instance in RAPID.</p>
Declared in module	<p>Select the module where the data and the part procedure will be declared. It is possible to create a new module for the part.</p> <p>If a task list is used, the module will be created in all tasks in the task list if it does not already exist, and the <code>partdata</code> and procedure will be placed in this module.</p> <p>Note: Only normal program modules will be visible in the list.</p>

Continues on next page

Default value	Description
Advanced part	Connect an advanced part to this part. See Example 1, advanced part on page 16 .

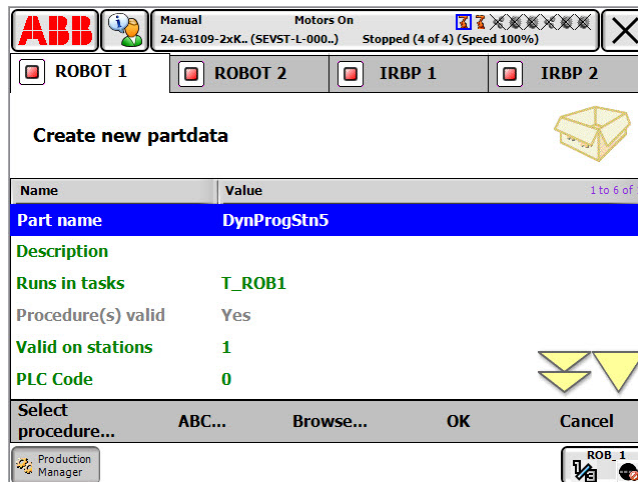
Create a new dynamic part

- 1 In the Production Manager main menu select **Part handling**.
- 2 Tap **New**.



xx1400002354

- 3 Select **Part name** in list and tap **ABC...**



xx1400002357

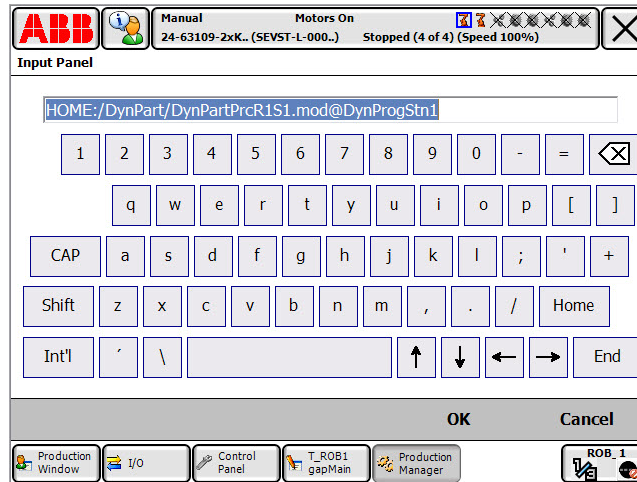
Continues on next page

2 Production Manager user interface

2.5.2 Create a new part

Continued

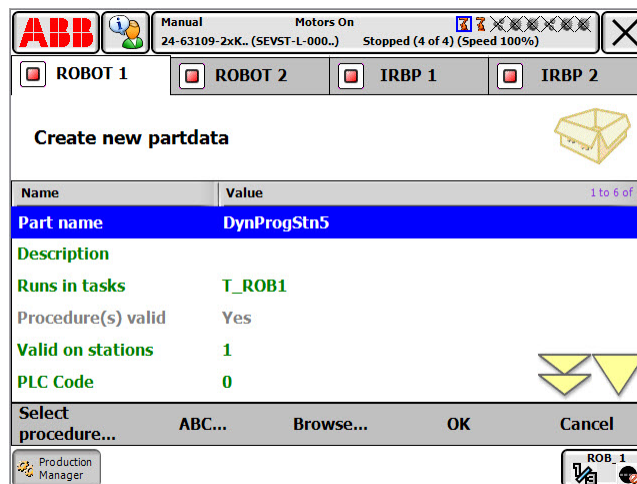
- 4 Enter the path to the module and procedure name separated with @. If module does not exist, it will be created.



xx1400002358

- 5 Or

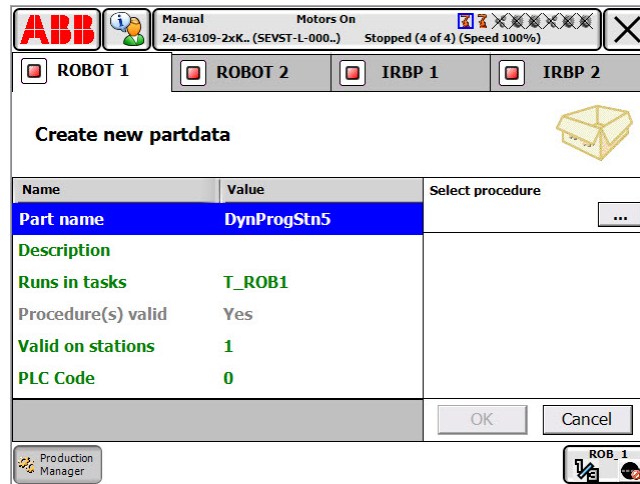
Select Part name in list and tap **Browse...**



xx1400002359

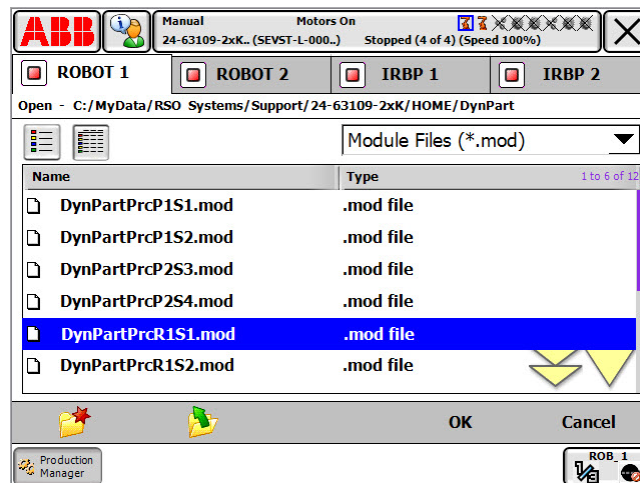
Continues on next page

6 Tap the button ... to browse for module.



xx1400002360

7 Select module.



xx1400002361

Continues on next page

2 Production Manager user interface

2.5.2 Create a new part

Continued

8 Select procedure and tap OK.

Name	Value	Select procedure
Part name	DynProgStn5	HOME:/DynPart/... [...]
Description		DynProgStn1
Runs in tasks	T_ROB1	
Procedure(s) valid	Yes	
Valid on stations	1	
PLC Code	0	

xx1400002362

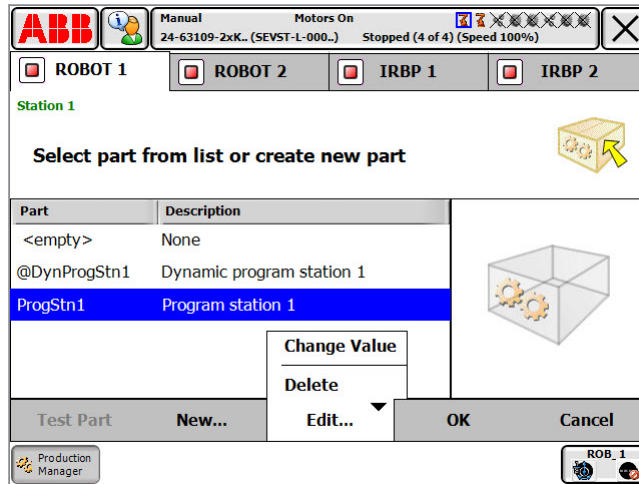
9 If part is synchronized in several tasks, select task in list and repeat step 4 or step 5 to 8.

Name	Value
Part name	HOME:/DynPart/DynPartPrCR1S1.mod@DynProgS...
Description	HOME:/DynPart/DynPartPrCR2S1.mod@DynProgS...
Runs in tasks	T_ROB1:T_ROB2:T_POS1
Procedure(s) valid	Yes

xx1400002363

2.5.3 Edit part

- 1 In the Production Manager main menu select **Part handling**.
- 2 Select a part to edit and tap **OK**.
- 3 Tap **Edit** and select **Change Value**.

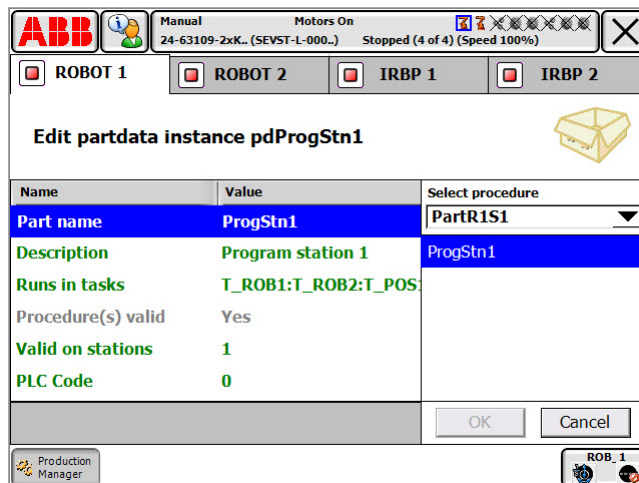


xx1400002364

- 4 The **Edit part** dialog has a number of fields to enter. See [Edit partdata dialog on page 48](#).

The user interface will help the user to create a new `partdata` instance together with the actual part procedure to be called during production.

Some of the fields in the dialog are loaded with default values and are not possible to edit.



xx1400002365

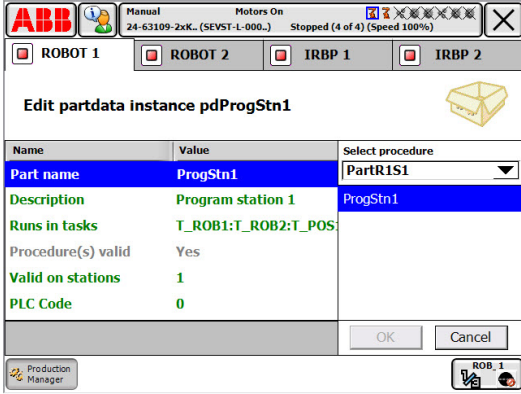
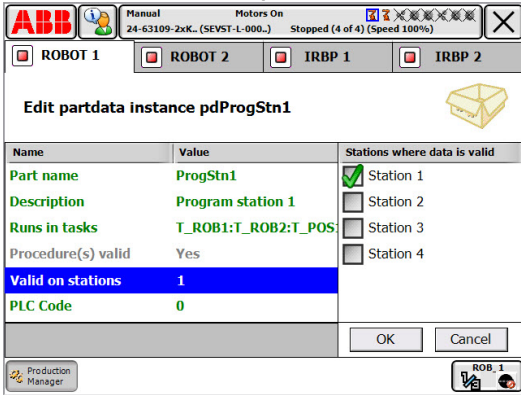
Continues on next page

2 Production Manager user interface

2.5.3 Edit part

Continued

Edit partdata dialog

Default value	Description
Part name	<p>This is the procedure that will be called during production. It is not possible to specify a new part name, only select from an already existing part.</p> <p>Use the dropdown list to search for procedures in another module.</p> 
Description	A custom string that describes the part.
Run in tasks	<p>These are the tasks this part should be declared in.</p> <p>Select the line and check the boxes in the window that appears on the right side.</p> <p>If more than one task is selected, it means that these tasks will be executed simultaneous.</p>
Procedure valid	This field cannot be edited. It indicates if the Part name and partdata instance name are valid in all tasks in the task list.
Valid on stations	<p>Select the stations this part will be valid on.</p> 
Partdata instance	The name of the partdata instance in RAPID. This field is not possible to edit.
Declared in module	The module where the data and the part procedure is declared. This field is not possible to edit.
Advanced part	Connect an advanced part to this part. See Example 1, advanced part on page 16 .

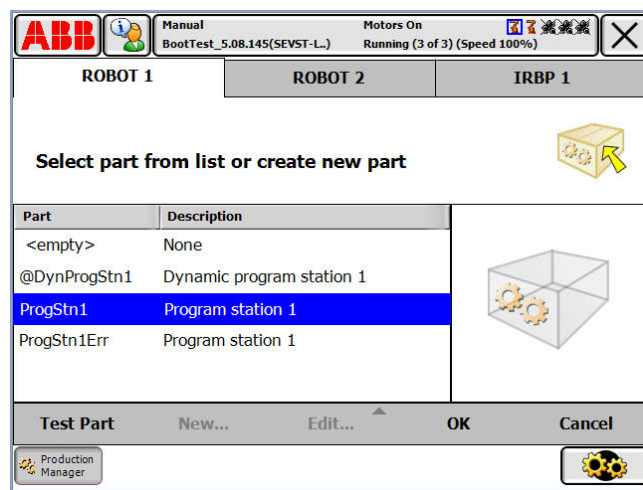
2.5.4 Test Part

The part testing functionality provides a way to test your parts without having to run the full production environment.

- No events in the production loop will be executed, thus it is useful to create custom service menus in the Production Manager to control the clamping etc, before and after running a test part.
- The part is only allowed to be tested at its valid station(s).
- Test Part is only allowed to run in manual mode.

Starting Test Part

- 1 Select a part in the table and tap Test Part.



xx1400002367



Note

The execution state must be in running mode to test the part.

2 Production Manager user interface

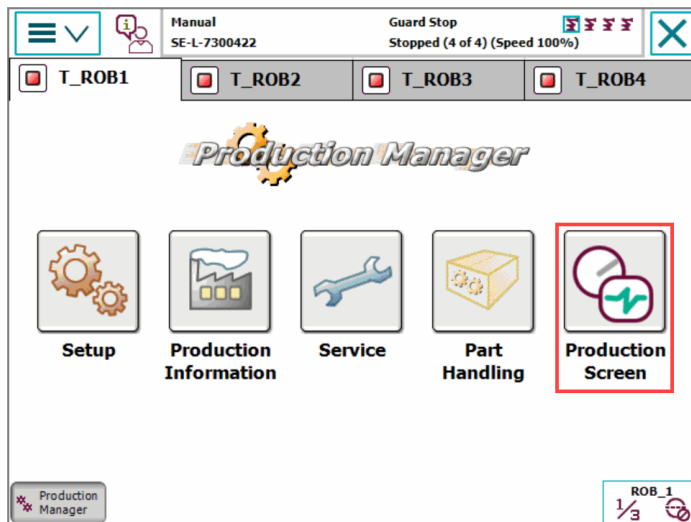
2.6 Custom application window

2.6 Custom application window

Launch application

If the system is loaded and configured with, for example, Production Screen, the application can be launched from the Production Manager desktop by clicking on the application icon to the right.

The application will be launched as a separate FlexPendant application outside Production Manager.






xx1400002332

2.7 State icons

Overview

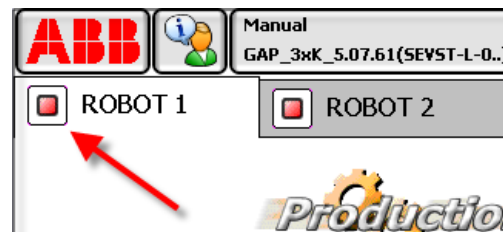
The Production Manager user interface uses state icons in order to display the state of the execution engine. Four different states are available.

States

State/Icon	Description
Running	No icon when the execution engine is running. This is the normal state.
Stopped  xx1400002369	The task has stopped.
Busy  xx1400002370	The task is producing a part or running a menu.
Blocked  xx1400002371	The task is blocked by another task currently running a Setup or Service menu. See Menudata on page 15 .

Location of state icons

The state icons are located at each tab.



xx1400002372



Note

Only normal tasks are visible as tabs in the user interface.

This page is intentionally left blank

3 Configuring Production Manager

3.1 Production Manager Task configuration

Overview

In *Production Manager Task* configuration it is possible to specify the look and feel of the Production Manager user interface.

System parameter	Description
<i>Task name</i>	Name of the task for witch the configuration is valid for.
<i>Tab text name</i>	Enter the name that should appear on the select in Production Manager.
<i>Tab index</i>	Specify the select order for the task. If the select index is not entered the selects can be added in an arbitrary order.
<i>Hide Production Info</i>	If Yes, the Production Information icon will be hidden.
<i>Hide Part Handler</i>	If Yes, the Part Handler icon will be hidden.

Type name	Task name	Tab text name	Tab index	Hide Production Info	Hide Part Handler
GAP Process Settings	T_POS1	IRBP 1	4	No	No
GAP Task	T_ROB1	ROBOT 1	1	No	No
GAP API State	T_ROB2	ROBOT 2	2	No	No
GAP API Commands	T_ROB3	ROBOT 3	3	No	No

xx1400002373

Production Manager Process Settings

Production Manager Process Settings specifies the application to be launched from Production Manager.

System parameter	Description
<i>Task name</i>	Name of the task for witch the configuration is valid for
<i>FlexPendant app dll</i>	The name of the dll to launch.
<i>FlexPendant app namespace</i>	The namespace of the dll to launch.
<i>FlexPendant app class name</i>	The class name of the dll to launch.
<i>Button-up image</i>	The name of the icon that will be displayed on Production Manager's desktop.
<i>Button-down image</i>	The name of the icon that will be displayed on Production Manager's desktop.
<i>Application name</i>	The name of the application. This name will be displayed on Production Manager's desktop.

Type name	Task name	FlexPendant app dll	FlexPendant app namespace	FlexPendant app class name
GAP Process Settings	T_ROB3	TpsViewArc.dll	ABB.Robotics.Processes.Arc	ArcViewDesktop
GAP Task	T_ROB2	TpsViewArc.dll	ABB.Robotics.Processes.Arc	ArcViewDesktop
GAP API State	T_ROB1	TpsViewArc.dll	ABB.Robotics.Processes.Arc	ArcViewDesktop

xx1400002374

Continues on next page

3 Configuring Production Manager

3.1 Production Manager Task configuration

Continued

Type name	Button up-image	Button down-image	Application name
GAP Process Settings	arcMainArcWelding96up.gif	arcmainarcWelding96down.gif	RobotWare Arc
GAP Task	arcMainArcWelding96up.gif	arcmainarcWelding96down.gif	RobotWare Arc
GAP API State	arcMainArcWelding96up.gif	arcmainarcWelding96down.gif	RobotWare Arc
GAP API Commands	arcMainArcWelding96up.gif	arcmainarcWelding96down.gif	RobotWare Arc

xx1400002375

Production Manager API State

Production Manager API State specifies the signals for tasks state.

System parameter	Description
<i>Task name</i>	Name of the task for witch the configuration is valid for.
<i>At-Safe DI</i>	The input signal name that specifies that the task is/is at safe. When this signal is high, it is considered safe to run execution of specific tasks.
<i>At-Service DI</i>	The input signal name that specifies that the task is at service. When this signal is high, it is considered safe to run execution of specific tasks.
<i>Running out signal</i>	Digital output signal specifying that <code>Prod Mgr</code> task is running.
<i>Ready out signal</i>	Digital output signal specifying that <code>Prod Mgr</code> task is ready for new order.
<i>Error group out signal</i>	Group output signal for error codes. If not configured, the error code will be mirrored to the PLC group output signal instead. PLC codes ≤ 99 can be used if configured.
<i>Error strobe out signal</i>	If defined this signal will go high when error occurs.
<i>Error ack in signal</i>	When set high, this signal will reset the error group output signal and strobe. If no ack is used, the error code will remain on the error group output signal (if defined).

Type name	Task name	At-Safe DO	At-Service DO
GAP Process Settings	T_POS1	siAtSafe	siAtService
GAP Task	T_ROB1	siAtSafeR1	siAtServiceR1
GAP API State	T_ROB3	siAtSafeR3	siAtServiceR3
GAP API Commands	T_ROB2	siAtSafeR2	siAtServiceR2

xx1400002376

In the example above, the positioner task T_POS1 is configured safe when all robot tasks, T_ROB1 T_ROB2 T_ROB3, are safe with cross connections. The same applies for service.

Production Manager API Commands

Production Manager API Commands specifies the signal interface for executing part and knowledge of at which station a task/robot is at and next station to go when order to run part is given.

System parameter	Description
<i>Task name</i>	Name of the task for witch the configuration is valid for.
<i>Run part in signal</i>	Input signal for running a part. Works on both PLC and Operator Ready interface.
<i>Run menu in signal</i>	Input signal for running a menu. Designed to use with PLC interface (PLC code required on <code>menudata</code>)

Continues on next page

3 Configuring Production Manager

3.1 Production Manager Task configuration

Continued

System parameter	Description
<i>Run ack signal</i>	Acknowledge signal used for handshake.
<i>Run ack timeout</i>	Defines the timeout when waiting for the run part or run menu signal to go low.
<i>PLC group in signal</i>	Group input signal that defines the PLC order.
<i>Allow 0 value for PLC</i>	Flag for allowing 0 value for PLC. If set, search for <code>partdata</code> with value 0 for PLC is done. Only used if PLC group in is configured.
<i>PLC group out signal</i>	Group output signal that confirms the PLC order. Also works as error code if an error occurs in Production Manager if no separate error signals are defined in <i>Production Manager API State</i> .
<i>No reset of PLC out signal</i>	Flag for specifying if PLC should be reset after part go signal. TRUE means no reset of PLC out signal. Default value is FALSE.
<i>At station 1 insignal</i>	Input signal that specifies that robot/task is at station 1.
<i>At station 2 insignal</i>	Input signal that specifies that robot/task is at station 2.
<i>At station 3 insignal</i>	Input signal that specifies that robot/task is at station 3.
<i>At station 4 insignal</i>	Input signal that specifies that robot/task is at station 4.
<i>At station 5 insignal</i>	Input signal that specifies that robot/task is at station 5.
<i>At station 6 insignal</i>	Input signal that specifies that robot/task is at station 6.
<i>At station 7 insignal</i>	Input signal that specifies that robot/task is at station 7.
<i>At station 8 insignal</i>	Input signal that specifies that robot/task is at station 8.
<i>Station 1 next insignal</i>	Input signal that specifies next station 1 for robot/task.
<i>Station 2 next insignal</i>	Input signal that specifies next station 2 for robot/task.
<i>Station 3 next insignal</i>	Input signal that specifies next station 3 for robot/task.
<i>Station 4 next insignal</i>	Input signal that specifies next station 4 for robot/task.
<i>Station 5 next insignal</i>	Input signal that specifies next station 5 for robot/task.
<i>Station 6 next insignal</i>	Input signal that specifies next station 6 for robot/task.
<i>Station 7 next insignal</i>	Input signal that specifies next station 7 for robot/task.
<i>Station 8 next insignal</i>	Input signal that specifies next station 8 for robot/task.
<i>Check aborted part</i>	Flag for check if part is finished.

Type name	Task name	Run part in signal	At station 1 insignal	At station 2 insignal
GAP Process Settings	T_ROB1	siOPOK_GAP	diLS_1_INPOS	diLS_2_INPOS
GAP Task	T_POS1	siOPOK_GAP	diLS_1_INPOS	diLS_2_INPOS
GAP API State	T_ROB3	siOPOK_GAP	diLS_1_INPOS	diLS_2_INPOS
GAP API Commands	T_ROB2	siOPOK_GAP	diLS_1_INPOS	diLS_2_INPOS

xx140002377

Continues on next page

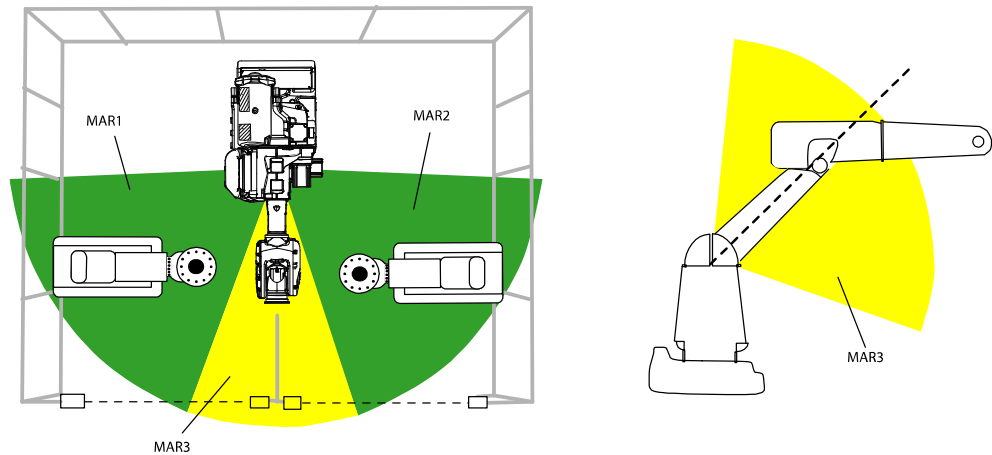
3 Configuring Production Manager

3.1 Production Manager Task configuration

Continued

Using signals from EPS

When using *Electronic Position Switches (EPS)* it is possible to setup signals so that Production Manager knows in which station the robot is and where the next station is.



xx0700000442

The below example is what needs to be added in the *EIO.cfg* for a setup with two positioners that are not indexing.

```
-Res "siGap_AtStn_1" -Act1 "PSC1MAR1"  
-Res "siGap_AtStn_2" -Act1 "PSC1MAR2"
```

For more information about EPS, see *Application manual - Electronic Position Switches*.

Production Manager Current Part

Production Manager Current Part specifies the executing part and station for a robot/task.

System parameter	Description
<i>Task name</i>	Name of the task for which the configuration is valid for.
<i>Instance name</i>	Name of executing <code>partdata</code> instance.
<i>Station</i>	Executing station robot/task.

3.2 Production Manager MultiMove Support

General

MultiMove systems are supported by loading the *Execution Engine* into all motion tasks. The engines may be triggered to run independently whenever in a ready state. Simultaneous execution is possible by triggering multiple engines to run concurrently. A task-list field in `menudata` and `partdata` allows the user to synchronize execution between multiple tasks.

The events in the *Execution Engine* support MultiMove. The tasks defined in the `TaskList` field of the `ee_event` data will be executed synchronously and the events will be synchronized. Each synchronized task will wait until all tasks in the `TaskList` are ready executing the current event before moving on to the next event.

How to load Execution Engine

To use Production Manager in tasks other than the robot tasks or positioner tasks `T_POS1/T_POS2`, the *Execution Engine* needs to be loaded in that task and be configured (see previous chapters).

To load *Execution Engine* a `SYS` config file needs to be loaded.

Copy the following configuration and replace `<taskname>` with actual TASK NAME.

```
SYS:CFG_1.0:5:0::
# CAB_EXEC_HOOKS:
#
-Routine "GapEE_PwrOnShelf" -Shelf "POWER_ON" -Task "<taskname>"
-Routine "GapEE_QStopShelf" -Shelf "QSTOP" -Task "<taskname>"
-Routine "GapEE_ResetShelf" -Shelf "RESET" -Task "<taskname>"
-Routine "GapEE_RestaShelf" -Shelf "RESTART" -Task "<taskname>"
-Routine "GapEE_StartShelf" -Shelf "START" -Task "<taskname>" -SeqNo
    100
-Routine "GapEE_StopShelf" -Shelf "STOP" -Task "<taskname>"
#

CAB_TASK_MODULES:
#
-File "RELEASE:/options/gap/GapCore/Code/GAP_ACCESS.sys" -Install
    -Task "<taskname>"
-File "RELEASE:/options/gap/GapCore/Code/GAP_SYNC.sys" -Install
    -Task "<taskname>"
-File "RELEASE:/options/gap/GapCore/Code/GAP_EE_EVT.sys" -Install
    -Task "<taskname>"
-File "RELEASE:/options/gap/GapCore/Code/GAP_EE.sys" -Install -Task
    "<taskname>"
-File "RELEASE:/options/gap/GapCore/Code/GAP_EVT.sys" -Install
    -Task "<taskname>"
-File "HOME:/GAP_USER.sys" -Task "<taskname>"
```

3 Configuring Production Manager

3.3 User Authorization System settings



3.3 User Authorization System settings

Defining access levels

Production Manager publishes a set of application grants that can be used to control the access to different functions within the application. Most application grants in Production Manager requires some controller grants, i.e activating all application grants for Production Manager does not automatically give access to all functionality within Production Manager, see grants table below.

If the logged on user has the controller grant **Full Access** it overrides all Production Manager application grants. That is, the application grants will automatically be true if **Full Access** is true.

The application grants can be found in the **UAS Administration Tool** in RobotStudio.

Application Grant	Description
Select Parts	If true, the user is allowed to select parts in stations in the Part Handling window. Requires the controller grant Modify current value .
Edit Parts	If true, the user is allowed to create, edit and delete parts. Requires the application grant Select Parts and the controller grant Edit RAPID code .
Debug Parts	If true, the user is allowed to run Production Manager parts in debug mode. Requires the controller grant Modify current value and I/O write access .
Edit Menus	Valid for both setup and service. If true, the user is allowed to create, edit and delete menus. Requires the controller grant Edit RAPID code .
Run Menu User Level	This grant level is connected to the byte <code>minUserLevel</code> field in the <code>menudata</code> . The logged on user is allowed to run the menu if this grant is true and the <code>minUserLevel</code> field in the <code>menudata</code> \leq Run Menu User Level. Min value: 0. Max value: 255. Requires the controller grant Modify current value and I/O write access .  Note If the controller grant Full Access is true, the Run Menu User Level grant will be true with value 0.
Run Seam Displacement	If this grant is true, the user is allowed to launch Seam Displacement from the Part Handling window. Requires the controller grant Perform ModPos and HotEdit .  Note Requires the RobotWare Arc option <i>Seam Displacement</i> .

Example: UAS settings for running menus

In this example the logged on user should be allowed to run the service routine `Move robot to safe position`, but not allowed to run the setup menu `Set/Change robot safe position`. Start by defining the application grant for the logged on user's group. In this example we set the threshold value for the **Run Menu User Level** to 40, see the following figure.

Continues on next page

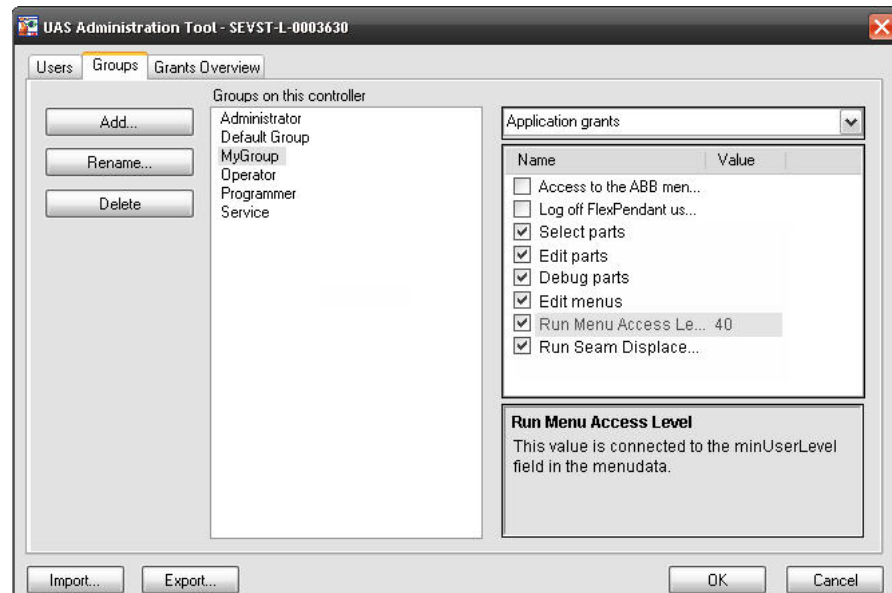
Now the two different menudata instances must be defined with different values in the minUserLevel field.

```
TASK PERS menudata mdRobSafe1:=["Move robot to safe position",  
    "GapIRB140Icon.gif", "CheckSafePos1", 255, "", 255, TRUE, 2, 20, FALSE, 0];
```

```
TASK PERS menudata mdRobSetSafe1:=["Set/Change robot safe position",  
    "GapIRB140Icon.gif", "SetSafePos1", 255, "", 255, TRUE, 1, 60, FALSE, 0];
```

The service menu Move robot to safe position has minUserLevel set to 20, that is, below the user's threshold value 40 specified in UAS Administration Tool, and is therefore allowed to be executed by the user.

The setup menu Set/Change robot safe position has minUserLevel set to 60, that is, larger than the user's threshold value for the Run Menu User Level grant, and is therefore not allowed to be executed by the user.



xx140002378





This page is intentionally left blank

4 Production Manager PLC support

4.1 How to run Production Manager from PLC

General

Production Manager can be controlled by using a PLC instead of the FlexPendant. By configuring the signals described in the table below, Production Manager can react directly on PLC orders used for running parts or menus. When Production Manager receives a PLC order, it searches the task for a `partdata` or `menudata` instance, depending on the order type, where the `PlcCode` field matches the value of the group input signal `plc_cmd_group_in`.

Task name	Run part in signal	Run menu in signal	Run ack signal	PLC group in signal	PLC group out signal
 T_POS1	siGap_Run_Part	diRunMenuP1	doPlcAckP1	giPlcCmd	goPlcCmd
 T_ROB1	siGap_Run_Part	diRunMenuR1	doPlcAckR1	giPlcCmd	goPlcCmd
 T_ROB3	siGap_Run_Part	diRunMenuR3	doPlcAckR3	giPlcCmd	goPlcCmd
 T_ROB2	siGap_Run_Part	diRunMenuR2	doPlcAckR2	giPlcCmd	goPlcCmd

xx1400002379

Description of signals

System parameter	Description
Run part in signal	Input signal for running a part
Run menu in signal	Input signal for running a menu.
Run ack signal	Acknowledge signal used for handshake.
PLC group in signal	Group input signal that defines the PLC order.
PLC group out signal	Group output signal that confirms the PLC order. Also works as error code if an error occurs in Production Manager.

Example workflow

The following is a typical workflow when running Production Manager from a PLC.

- 1 The PLC sets an order on the PLC group in signal.
- 2 The order is confirmed by Production Manager by setting the PLC group output signal to the same value as the PLC group in signal.
- 3 The PLC sets the *Run part* or *Run menu* signal.

If everything is working correctly, Production Manager will set the PLC Group output signal to 0.



Note

If *No reset of output* signal in GAP API State is TRUE, the PLC value will remain on the group output signal.

Continues on next page

4 Production Manager PLC support

4.1 How to run Production Manager from PLC

Continued

If an error has occurred then the last two digits in the error code will be set on the PLC group output signal, that is `error_code - 111400` since Production Manager's error codes works between 111400 and 111499.



Note

If *Error group output signal* in GAP API State is specified, the error code will not be displayed on the PLC group output signal, but instead on the *Error group output signal*. See [Production Manager API State on page 54](#).

- 4 If the *Run ack* signal has been defined further handshaking is possible to use before the part or menu is executed.

Assuming everything is working correctly, *Run ack* signal will be set high by Production Manager.

- 5 The PLC responds with setting the *Run part* or *Run menu* signal low which will trigger Production Manager to set the *Run ack* signal low again and launch the part or menu.



Note

Due to the error code functionality described above we recommend that the PLC orders do not use numbers between 0 and 99.

4.2 How to run Production Manager from PLC via RAPID

Overview

Sometimes the PLC logic needs to be processed in a RAPID module before it is served to Production Manager. For these occasions Production Manager provides an instruction interface in RAPID from where it is possible to tell Production Manager which procedure to execute. *Parts*, *Setup*, and *Service* menus can be run by serving Production Manager the type of order and procedure to execute.

How to run parts from PLC via RAPID

Example 1

```
PERS partdata pdProgStn1:["ProgStn1", "Program station 1",
  "T_ROB1:T_ROB2:T_ROB3:T_POS1",1, "GapEmptyPart200.gif",""];
PERS partdata pdProgStn2:["ProgStn2","Program station 2",
  "T_ROB1:T_ROB2:T_ROB3:T_POS1",2,"GapEmptyPart200.gif",""];

CONNECT inPlcCmd WITH trPlcCmd; ISignalDI diPlcCmd,1, inPlcCmd

LOCAL TRAP trPlcCmd
VAR num nPlcCode;
nPlcCode:= Ginput(giPlcCode);
TEST nPlcCode
CASE 1:
  SetDO soGap_NextStn1,0;
  ! run part for station 1
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_ROB1"),1,pdProgStn1;
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_ROB2"),1,pdProgStn1;
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_ROB3"),1,pdProgStn1;
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_POS1"),1,pdProgStn1;
  ! tell GAP next station
  SetDO soGap_NextStn1,1;
CASE 2:
  SetDO soGap_NextStn2,0;
  ! run part for station 2
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_ROB1"),2,pdProgStn2;
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_ROB2"),2,pdProgStn2;
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_ROB3"),2,pdProgStn2;
  PMgrSetNextPart \
    TaskNumber:=GapTaskIndex("T_POS1"),2,pdProgStn2;
  ! tell GAP next station
  SetDO soGap_NextStn2,1;
CASE 12:
CASE 64:
```

Continues on next page

4 Production Manager PLC support

4.2 How to run Production Manager from PLC via RAPID

Continued

```
CASE 128:
ENDTEST
! tell GAP to run
SetDO soGap_Run,1;
ENDTRAP
```

How to run menus from PLC via RAPID

Example 2

```
TASK PERS menudata mdCalibIntch1:=["Calibrate irbp1 interchange
positions", "GapMicCalibrate32.gif", "Irbp1Mnu:mnuCalibIntch1",
255, "", 3, TRUE, 1, 0, TRUE, 0];

CONNECT inPlcCmd WITH trPlcCmd;
ISignalDI diPlcCmd,1, inPlcCmd;

LOCAL TRAP trPlcCmd
VAR num nPlcCode;
VAR menudata mdTemp;

nPlcCode:= Ginput(giPlcCode);
TEST nPlcCode
CASE 1:
mdTemp:= mdCalibIntch1;
PMgrRunMenu mdTemp;
ENDTEST
ENDTRAP
```

To run a menu in several tasks

Example 3

To run a menu in several tasks the following instructions can be used:

```
LOCAL CONST menudata mdRobSafeAll:=["Move all robots to home
position", "GapIRB140Icon.gif", "MoveSafe", 255, "
T_ROB1:T_ROB2:T_ROB3",
GAP_SHOW_ALWAYS, TRUE, GAP_SERVICE_TYPE, 0, FALSE, 0];
```

```
VAR menudata mdTemp;
```

```
mdTemp:=mdRobSafeAll;
PMgrRunMenu \ TaskNumber:=GapTaskIndex("T_ROB1"), mdTemp;
PMgrRunMenu \ TaskNumber:=GapTaskIndex("T_ROB2"), mdTemp;
PMgrRunMenu \ TaskNumber:=GapTaskIndex("T_ROB3"), mdTemp;
```

Or:

```
TASK PERS menudata mdRobSafeAll:= ["Move all robots to home
position", "GapIRB140Icon.gif", "MoveSafe", 255, "
T_ROB1:T_ROB2:T_ROB3",
GAP_SHOW_ALWAYS, TRUE, GAP_SERVICE_TYPE, 0, FALSE, 0];
```

Continues on next page


```
PMgrRunMenu \ TaskNumber:=GapTaskIndex("T_ROB1"), mdRobSafeAll;  
PMgrRunMenu \ TaskNumber:=GapTaskIndex("T_ROB2"), mdRobSafeAll;  
PMgrRunMenu \ TaskNumber:=GapTaskIndex("T_ROB3"), mdRobSafeAll;
```

This page is intentionally left blank

5 RAPID references

5.1 Instructions

5.1.1 ExecEngine - Start execution engine

Usage

`ExecEngine` starts the execution engine.

Basic examples

The following example illustrates the instruction `ExecEngine`.

Example 1

```
ExecEngine;
```

The execution engine is started and waiting for an order.

Arguments

There are no arguments.

Program execution

The user calls this routine from the main routine in each motion task. Typically the user-defined main routine should have a procedure call to `ExecEngine` and nothing else.

Syntax

```
ExecEngine ';' ;
```

5 RAPID references

5.1.2 PMgrGetNextPart - Get active part for station in task

5.1.2 PMgrGetNextPart - Get active part for station in task

Usage

`PMgrGetNextPart` gets the part that is being produced for a station in a task.

Basic examples

The following examples illustrate the instruction `PMgrGetNextPart`.

Example 1

```
PMgrGetNextPart stn, tmpPart;
```

The `PMgrGetNextPart` instruction will return the `partdata` used for station `stn`. The resulting `partdata` will be passed in `tmpPart`.

Example 2

```
! Data declarations
VAR num station:=1;
VAR partdata pdTmpChk;
VAR string sPartDataName;
PROC PrepareData()
    PMgrGetNextPart station,pdTmpChk\InstanceName:=sPartDataName;
ENDPROC
```

Arguments

```
PMgrGetNextPart [\TaskNumber] Station RetData [\InstanceName]
```

`[\TaskNumber]`

Data type: num

Optional argument specifying the Production Manager specific task number to get the part for. If argument `TaskNumber` is omitted the current task number is used.

`Station`

Data type: num

The station to get the part for.

`RetData`

Data type: partdata

The resulting part returned from the instruction.

`[\InstanceName]`

Data type: string

The instance name of the `retData` part.

Program execution

The instruction returns the part selected for the specified station and task number. In the case no parts are selected for the station(s), for example, running Production Manager from a PLC, an empty part will be returned.

Continues on next page

5.1.2 PMgrGetNextPart - Get active part for station in task *Continued*

Syntax

```
PMgrGetNextPart
  [ '\ ' TaskNumber ':=' ] < expression (IN) of num > ','
  [ Station ':=' ] < expression (IN) of num > ','
  [ RetData ':=' ] < var or pers (INOUT) of partdata > ','
  [ InstanceName ':=' ] < var or pers (INOUT) of string > ';'
;
```

5 RAPID references

5.1.3 PMgrSetNextPart - Set active part for station in task

5.1.3 PMgrSetNextPart - Set active part for station in task

Usage

`PMgrSetNextPart` sets the part that will be produced for a station in a task.

Basic examples

The following examples illustrate the instruction `PMgrSetNextPart`.

Example 1

```
PMgrSetNextPart stn, tmpPart
```

The `SetNextPart` instruction will set the part `tmpPart` for station `stn`.

Example 2

```
! Data declarations
VAR num station:=1;
VAR partdata pdTmpChk;

PROC PrepareData ()
  PMgrSetNextPart station,pdTmpChk;
ENDPROC
```

Arguments

```
PMgrSetNextPart [ \TaskNumber ] Station NewData
```

[\TaskNumber]

Data type: num

Optional argument specifying the Production Manager specific task number to set the part for. If argument `TaskNumber` is omitted the current task number is used.

Station

Data type: num

The station to set the part for.

NewData

Data type: partdata

The part that should be set for this station and task.

Program execution

The instruction sets the part for the specified station and task number.

Syntax

```
PMgrSetNextPart
  [ '\ ' TaskNumber ' := ' ] < expression (IN) of num > ', '
  [ Station ' := ' ] < expression (IN) of num > ', '
  [ NewData ' := ' ] < persistent (PERS) of partdata > ', '
```

5.1.4 PMgrRunMenu - Run menu in task

Usage

PMgrRunMenu is used to run a menu in a task.

Basic examples

The following examples illustrate the instruction PMgrRunMenu.

Example 1

```
VAR menudata mnuBE := ["TCP Setup", "", "BEToolSetup", 255, "",
    GAP_SHOW_ALWAYS, TRUE, GAP_SETUP_TYPE, 0, FALSE, 0];
PMgrRunMenu mnuBE;
```

Runs the mnuBE menu in the current task, without using the FlexPendant application.

Example 2

```
! Data declarations
VAR menudata mnuBE := ["TCP
    Setup", "", "BEToolSetup", 255, "", GAP_SHOW_ALWAYS,
    TRUE, GAP_SETUP_TYPE, 0, FALSE, 0];
VAR num taskNr;
PROC Procl ()
    taskNr := GAP_TASK_NO;
    PMgrRunMenu(\TaskNumber:=taskNr, mnuBE);
ENDPROC
```

Arguments

PMgrRunMenu [\TaskNumber] Menu

[\TaskNumber]

Data type: num

Optional argument specifying the Production Manager specific task number to run the menu in. If argument TaskNumber is omitted, the current task number is used.

Menu

Data type: menudata

The menu to execute.

Syntax

```
PMgrRunMenu
[ '\ ' TaskNumber := ' < expression (IN) of num > ]
[ Menu := ' ] < var or pers (INOUT) of menudata > ';' ;
```

5 RAPID references

5.2.1 PMgrAtSafe - Check if task is at safe state

5.2 Functions

5.2.1 PMgrAtSafe - Check if task is at safe state

Usage

PMgrAtSafe is used to check if the task is at safe state.

Basic examples

The following examples illustrate the function PMgrAtSafe.

Example 1

```
VAR bool bAtSafe;  
bAtSafe:=PMgrAtSafe();
```

Check if the current task is at safe.

Example 2

```
! Data declarations  
VAR bool bAtSafe;  
VAR num taskNr;  
PROC Procl ()  
    taskNr := GAP_TASK_NO;  
    bAtSafe:=PMgrAtSafe(\TaskNumber:=taskNr);  
ENDPROC
```

Return value

Data type: bool

TRUE if the task is at safe state, FALSE otherwise.

Arguments

PMgrAtSafe [\TaskNumber]

[\TaskNumber]

Data type: num

Optional argument specifying the Production Manager specific task number to check the safe state for. If argument TaskNumber is omitted, the current task number is used.

Syntax

```
PMgrAtSafe '('  
    [ '\TaskNumber' := ' ] < expression (IN) of num > ')'
```

5.2.2 PMgrAtService - Check if task is at service state

Usage

PMgrAtService is used to check if task is at service state.

Basic examples

The following examples illustrate the function PMgrAtService.

Example 1

```
VAR bool bAtService;
bAtService:=PMgrAtService();
```

Check if the current task is at service.

Example 2

```
! Data declarations
VAR bool bAtService;
VAR num taskNr;
PROC Procl ()
  taskNr := GAP_TASK_NO;
  bAtService:=PMgrAtService(\TaskNumber:=taskNr);
ENDPROC
```

Return value

Data type: bool

TRUE if the task is at service state, FALSE otherwise.

Arguments

PMgrAtService [\TaskNumber]

[\TaskNumber]

Data type: num

Optional argument specifying the Production Manager specific task number to check the service state for. If argument TaskNumber is omitted, the current task number is used.

Syntax

```
PMgrAtService '('
  [ '\ ' TaskNumber ':' '=' ] < expression (IN) of num > ')'
```

5 RAPID references

5.2.3 PMgrAtState - Check the state of a task

5.2.3 PMgrAtState - Check the state of a task

Usage

PMgrAtState is used to check production state of a task.

Basic examples

The following examples illustrate the function PMgrAtState.

Example 1

```
VAR num PMState;  
PMState:=PMgrAtState();
```

Get the production state of the current task.

Example 2

```
! Data declarations  
VAR num state;  
VAR num taskNr;  
PROC Procl ()  
    taskNr := GAP_TASK_NO;  
    state:=PMgrAtState(\TaskNumber:=taskNr);  
ENDPROC
```

Return value

Data type: num

The returned value represents different execution states of Production Manager's execution engine.

The following return values are valid:

Constant	Value	Description
GAP_STATE_UNKN	0	Unknown state/not running
GAP_STATE_IDLE	1	Executing but idle
GAP_STATE_SETUP	2	Executing setup routine
GAP_STATE_PART	3	Executing part
GAP_STATE_SERV	4	Executing service routine

Arguments

PMgrAtState [\TaskNumber]

[\TaskNumber]

Data type: num

Optional argument specifying the Production Manager specific task number to get the state for. If argument TaskNumber is omitted, the current task number is used.

Syntax

```
PMgrAtState '('  
    [ '\TaskNumber :=' ] < expression (IN) of num > ')'
```

5.2.4 PMgrAtStation - Get the current station for a task

Usage

PMgrAtStation is used to get the current station for a task.

Basic examples

The following examples illustrate the function PMgrAtStation.

Example 1

```
VAR num nStation;
nStation:=PMgrAtStation();
```

Get the current station for the current task.

Example 2

```
! Data declarations
VAR num nStation;
VAR num taskNr;
PROC Procl ()
  taskNr := GAP_TASK_NO;
  nStation:=PMgrAtStation(\TaskNumber:=taskNr);
  TPWrite "Current station is" + ValToStr(nStation);
ENDPROC
```

Return value

Data type: num

The returned value represents the active station.

Arguments

PMgrAtStation [\TaskNumber]

[\TaskNumber]

Data type: num

Optional argument specifying the Production Manager specific task number to get the station for. If argument TaskNumber is omitted, the current task number is used.

Syntax

```
PMgrAtStation '('
  [ '\ ' TaskNumber ':' '=' ] < expression (IN) of num > ')'
```

5 RAPID references

5.2.5 PMgrNextStation - Get the next station for a task

5.2.5 PMgrNextStation - Get the next station for a task

Usage

PMgrNextStation is used to get the next station for a task.

Basic examples

The following examples illustrate the function PMgrNextStation.

Example 1

```
VAR num nextStation;  
nextStation:=PMgrNextStation();
```

Get the next station for the current task.

Example 2

```
! Data declarations  
VAR num nextStation;  
VAR num taskNr;  
PROC Procl ()  
    taskNr := GAP_TASK_NO;  
    nextStation:=PMgrNextStation(\TaskNumber:=taskNr);  
    TPWrite "Next station is" + ValToStr(nextStation);  
ENDPROC
```

Return value

Data type: num

The returned value represents the next station that will be used for the next part.

Arguments

```
PMgrNextStation [\TaskNumber]
```

[\TaskNumber]

Data type: num

Optional argument specifying the Production Manager specific task number to get the station for. If argument TaskNumber is omitted, the current task number is used.

Syntax

```
PMgrNextStation '('  
    [ '\ ' TaskNumber ':' '=' ] < expression (IN) of num > ')'
```

5.2.6 PMgrTaskNumber - Get the task number

Usage

`PMgrTaskNumber` is used to get the Production Manager specific task number. This task number is used in many instructions in the public RAPID interface of Production Manager.

Basic examples

The following example illustrates the instruction `PMgrTaskNumber`.

Example 1

```
VAR num taskNumber;
taskNumber := PMgrTaskNumber(\TaskName:="T_ROB1");
```

Get the Production Manager specific task index for task `T_ROB1`.

Return value

Data type: `num`

The returned value represents the Production Manager specific task index for the provided task name.

If no optional argument is used the task number for current task is returned.

Returns 0 if given `TaskName` is not a valid Production Manager task.

Arguments

```
PMgrTaskNumber [\TaskName]
```

`[\TaskName]`

Data type: `string`

The name of the task to get the task number for. If argument `TaskName` is omitted, the current task name is used.

Syntax

```
PMgrTaskNumber '('
  [ '\ ' TaskName ' := ' ] < expression (IN) of string > ')'
```

5 RAPID references

5.2.7 PMgrTaskName - Get the task name

5.2.7 PMgrTaskName - Get the task name

Usage

`PMgrTaskName` is used to get the task name connected to a Production Manager specific task number.

Basic examples

The following example illustrates the function `PMgrTaskName`.

Example 1

```
VAR string taskName;  
VAR num taskNumber;  
taskNumber:=GAP_TASK_NO;  
taskName:= PMgrTaskName (\TaskNumber:=taskNumber);  
TPWrite "The name of this task is " + taskName;
```

Get the task name for current task.

Return value

Data type: string

The returned value represents the task name connected to the Production Manager specific task number.

Arguments

`PMgrTaskName` [`\TaskNumber`]

[`\TaskNumber`]

Data type: num

The Production Manager specific task number. If argument `TaskNumber` is omitted, the current task number is used.

Syntax

```
PMgrTaskName '('  
[ \ ' TaskNumber ' := ' ] < expression (IN) of num > ')'
```

5.3 Public constants

Description

The list below shows the public constants and variables provided by Production Manager.

General

Public task specific constants:

Constant	Description
GAP_TASK_NO	The Production Manager specific task index for current task.
GAP_TASK_NAME	The task name of current task.

Menus

type field

Public constants to be used in the `type` field of `menudata` instances:

Constant	Value
GAP_SETUP_TYPE	1
GAP_SERVICE_TYPE	2

Example:

```
VAR menudata mnuBE := [ "TCP Setup", "", "BEToolSetup", 255, "",
    GAP_SHOW_ALWAYS, TRUE, GAP_SETUP_TYPE, 0, FALSE, 0 ];
```

validPosition field

Public constants to be used in the `validPosition` field of `menudata` instances:

Constant	Value
GAP_SHOW_NEVER	0
GAP_SHOW_SAFE	1
GAP_SHOW_SERVICE	2
GAP_SHOW_ALWAYS	255

Example:

```
VAR menudata mnuBE := [ "Check TCP", "", "BEToolCheck", 255, "",
    GAP_SHOW_NEVER, TRUE, GAP_SERVICE_TYPE, 0, FALSE, 0 ];
```

Execution

Execution state

Public constants to be used when querying Production Manager for the task state:

Constant	Value	Description
GAP_STATE_UNKN	0	Unknown state/not running
GAP_STATE_IDLE	1	Executing but idle
GAP_STATE_SETUP	2	Executing setup routine
GAP_STATE_PART	3	Executing part

Continues on next page

5 RAPID references

5.3 Public constants

Continued

Constant	Value	Description
GAP_STATE_SERV	4	Executing service routine

Example:

```
VAR num PMState;  
PMState:=AtState();  
IF PMState = GAP_STATE_IDLE THEN  
    TPWrite "Production Manager waiting for job";  
ENDIF
```

Events

Public constants to be used when defining events:

Constant	Value	Description
EE-START	1	Runs when exec engine starts
EE_CYCLE_START	2	Runs right after OP pressed/order from PLC
EE_PROC_START	3	Runs before menu executes
EE_PRE_PROD	4	General pre-production event
EE_CLOSE_JIG	5	Close jig
EE_INDEX	6	Index IRBP
EE_PRE_PART	7	Runs before part
EE_POST_PART	8	Runs after part
EE_OPEN_JIG	9	Open jig
EE_SERVICE	10	Run service of tool/other
EE_POST_PROD	11	General post-production event
EE_ABORT	12	Abort cycle
EE_WAIT_ORDER	13	Waiting for an order.
EE_POST_PROC	14	Runs after menu

6 Seam Displacement options

6.1 General

Overview

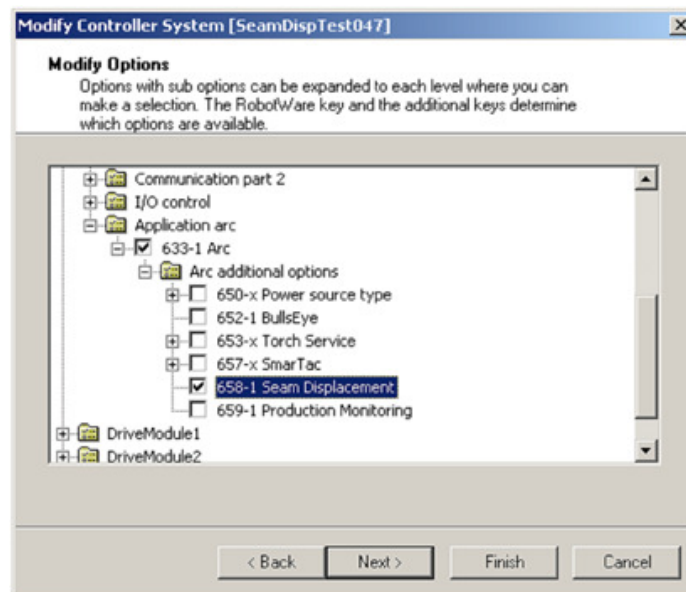
The Seam Displacement option allows the operator to shift seams in relation to a reference frame. The displacements can be applied via FlexPendant operator screens without stopping production. It is possible to shift an entire weld or targets within a seam individually. The operator can enter offsets at any point in time, whether the robot is welding or not. The applied changes will take effect in the next production cycle. This lets the operator visually inspect a part, apply seam offsets where needed, and the changes will take effect when the next part is welded.

User restrictions

The functions available in Seam Displacement may be restricted by the user authorization system, UAS.

About the option

The Seam Displacement is a separate Arc option.



xx140002380

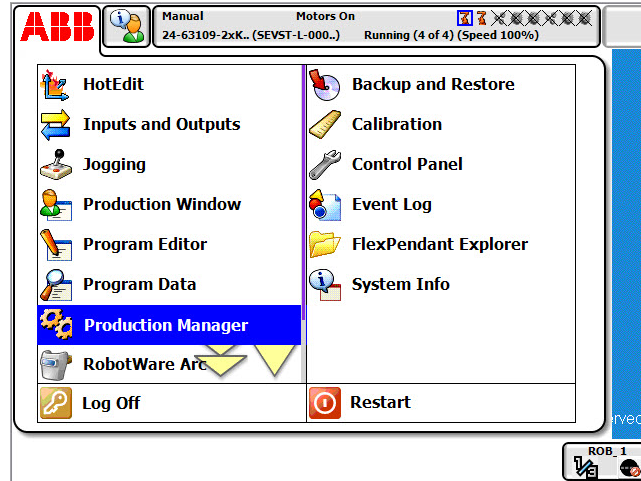
6 Seam Displacement options

6.2 Starting Seam Displacement option

6.2 Starting Seam Displacement option

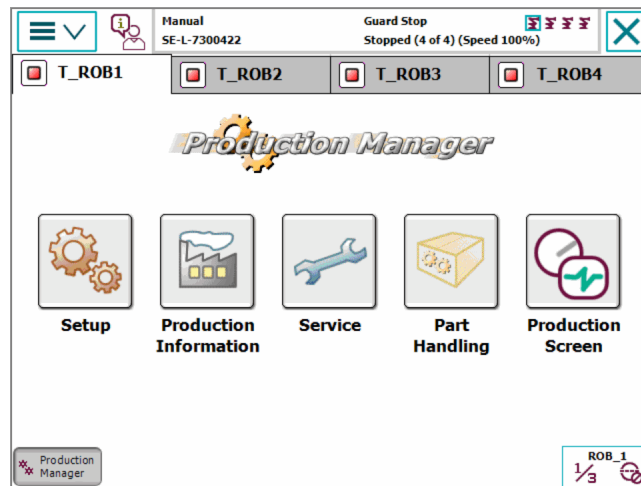
The seam displacement option is started as follows:

- 1 Go to the **ABB** menu and launch the **Production Manager**.



xx1400002330

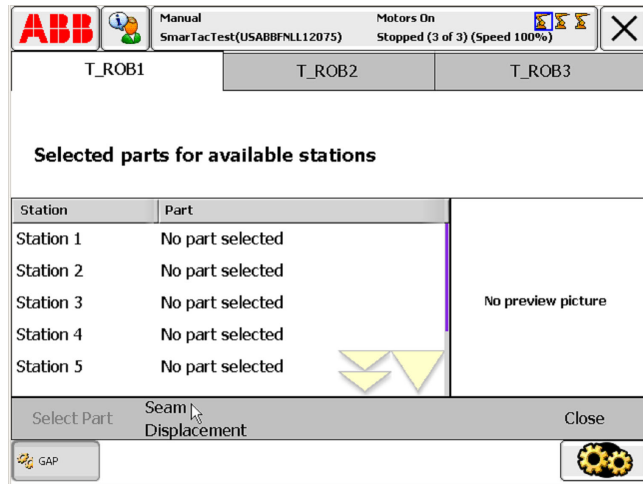
- 2 In the **Production Manager** main menu select **Part Handling**.



xx1400002331

Continues on next page

3 Tap Seam Displacement on the bottom menu.



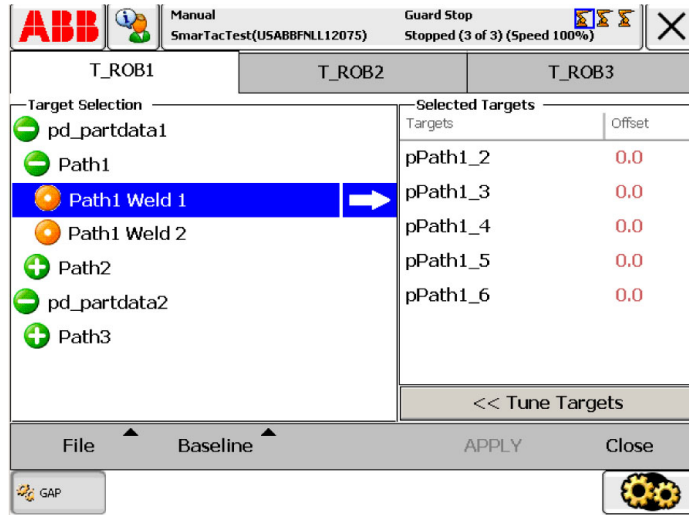
xx1400002381

6 Seam Displacement options

6.3 Functions available in Seam Displacement


6.3 Functions available in Seam Displacement

FlexPendant screen appearance



xx1400002401

Seam Displacement dialog

	Description
Target selection	Select welds from the tree view and add them to the right-hand section by selecting the arrow.  Note If a weld is used in more than one routine, it will appear the same everywhere it is used. Changes made to the offset will be the same for everywhere it is used.
Selected targets	Lists all selected targets within a weld and their current offset. Select the recycle bin to the right to remove the target from the selection.
File	You can save and load selections of often-used targets using the File menu. If your system uses UAS, this may be the only way to select targets for editing.
Baseline	To apply or reject the changes made to offset values, select: <ul style="list-style-type: none"> • Restore to original to discard all changes to the currently selected target positions • Restore entire program to original to discard all changes to target positions (also applies to changes made in the program editor) • Commit to current to apply all current changes to the selected target positions • Commit entire program to current to apply all changes to target positions (also applies to changes made in the program editor)
Tune targets	Tap Tune targets to display a keyboard for editing the offset values. The offset value is the length of the vector calculated from the x, y and z values changed in the Tune targets menu.
APPLY	Tap APPLY to apply changes made in the Tune targets menu.

Continues on next page

Related information

Positions can also be modified by jogging the robot to the new position.

This page is intentionally left blank

Index

A

API, 9

C

configuration, 53
 constants, 79
 create menu, 26
 create part, 41
 custom application window, 50

D

debug part, 37
 dynamic part, 43
 dynamic parts and menus, 16

E

edit menu, 32
 edit part, 47
 EE_ABORT, 80
 EE_CLOSE_JIG, 80
 EE_CYCLE_START, 80
 EE_INDEX, 80
 EE_OPEN_JIG, 80
 EE_POST_PART, 80
 EE_POST_PROC, 80
 EE_POST_PROD, 80
 EE_PRE_PART, 80
 EE_PRE_PROD, 80
 EE_PROC_START, 80
 EE_SERVICE, 80
 EE_WAIT_ORDER, 80
 EE-START, 80
 events, 12, 80
 ExecEngine, 67
 Execution Engine, 10
 execution state, 79

F

filter, 34
 functions, 72

G

GAP_SERVICE_TYPE, 79
 GAP_SETUP_TYPE, 79
 GAP_SHOW_ALWAYS, 79
 GAP_SHOW_NEVER, 79
 GAP_SHOW_SAFE, 79
 GAP_SHOW_SERVICE, 79
 GAP_STATE_IDLE, 79
 GAP_STATE_PART, 79
 GAP_STATE_SERV, 80
 GAP_STATE_UNKN, 79

GAP_TASK_NAME, 79
 GAP_TASK_NO, 79

I

icons, 51
 instructions, 67

M

main menu, 20
 menudata, 15
 MultiMove, 21, 57

N

new menu, 26
 new part, 41

P

partdata, 15
 Part handler, 37
 PLC support, 61
 PMgrAtSafe, 72
 PMgrAtService, 73
 PMgrAtState, 74
 PMgrAtStation, 75
 PMgrGetNextPart, 68
 PMgrNextStation, 76
 PMgrRunMenu, 71
 PMgrSetNextPart, 70
 PMgrTaskName, 78
 PMgrTaskNumber, 77
 preview, 39
 Production Information, 35
 Production Screen, 20
 public constants, 79

R

RAPID constants, 79
 RAPID functions, 72
 RAPID instructions, 67

S

Seam Displacement, 81
 Service menu, 24
 Setup menu, 22
 starting Production Manager, 19
 state icons, 51
 system parameters, 53

T

test part, 49

U

User Authorization System, 58
 user interface, 19



ABB AB

Robotics & Discrete Automation

S-721 68 VÄSTERÅS, Sweden

Telephone +46 (0) 21 344 400

ABB AS

Robotics & Discrete Automation

Nordlysvegen 7, N-4340 BRYNE, Norway

Box 265, N-4349 BRYNE, Norway

Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.

Robotics & Discrete Automation

No. 4528 Kangxin Highway

PuDong District

SHANGHAI 201319, China

Telephone: +86 21 6105 6666

ABB Inc.

Robotics & Discrete Automation

1250 Brown Road

Auburn Hills, MI 48326

USA

Telephone: +1 248 391 9000

abb.com/robotics