# 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

<table>
<thead>
<tr>
<th>References</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety manual for robot - Manipulator and IRC5 or OmniCore controller 1</td>
<td>3HAC031045-001</td>
</tr>
<tr>
<td>Introduction and Safety - Arc Welding Products</td>
<td>Introduction and Safety - Arc Welding Products</td>
</tr>
<tr>
<td>Operating manual - IRC5 with FlexPendant</td>
<td>3HAC050941-001</td>
</tr>
<tr>
<td>Operating manual - RobotStudio</td>
<td>3HAC050941-001</td>
</tr>
<tr>
<td>Operating manual - Getting started, IRC5 and RobotStudio</td>
<td>3HAC050941-001</td>
</tr>
<tr>
<td>Technical reference manual - RAPID Instructions, Functions and Data types</td>
<td>3HAC050917-001</td>
</tr>
<tr>
<td>Technical reference manual - RAPID Overview</td>
<td>3HAC050947-001</td>
</tr>
<tr>
<td>Technical reference manual - System parameters</td>
<td>3HAC050948-001</td>
</tr>
<tr>
<td>Application manual - Torch services</td>
<td>3HAC050981-001</td>
</tr>
</tbody>
</table>

Continues on next page
Overview of this manual

Continued

<table>
<thead>
<tr>
<th>References</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application manual - Production Screen</td>
<td>3HAC050964-001</td>
</tr>
</tbody>
</table>

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

Revisions

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Published with RobotWare 6.0.</td>
</tr>
<tr>
<td>A</td>
<td>Published with RobotWare 6.05.</td>
</tr>
<tr>
<td></td>
<td>• Added section Using signals from EPS on page 56.</td>
</tr>
<tr>
<td>B</td>
<td>Published with RobotWare 6.07.</td>
</tr>
<tr>
<td></td>
<td>• Updated section Production Manager MultiMove Support on page 57.</td>
</tr>
<tr>
<td>C</td>
<td>Published with RobotWare 6.13.</td>
</tr>
<tr>
<td></td>
<td>• Updated screenshots from FlexPendant.</td>
</tr>
</tbody>
</table>
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
1.2 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
external device such as PLC. The engine is capable of running part routines, setup routines, and service routines.
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

<table>
<thead>
<tr>
<th>ee_event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ee_eventnum action</td>
<td>The type of event to subscribe to. See list of available events in List of available events on page 12.</td>
</tr>
<tr>
<td>string procName</td>
<td>The actual procedure that will be called when running this event.</td>
</tr>
<tr>
<td>string taskList</td>
<td>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.</td>
</tr>
<tr>
<td>byte sortOrder</td>
<td>Value from 0-255 defining in which order this event will be executed with respect to other events of the same action type.</td>
</tr>
<tr>
<td>byte validStation</td>
<td>Value from 0-255 defining for which station this event is valid.</td>
</tr>
</tbody>
</table>

List of available events

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

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

Continued

Event order
1.4 Production Manager Menudata and Partdata

### Menudata

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string description</td>
<td>A short description of the menu item.</td>
</tr>
<tr>
<td>string image</td>
<td>The name of the icon to use next to the menu item.</td>
</tr>
<tr>
<td>string procName</td>
<td>The actual procedure that will be called when running this setup or service routine. The procName 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.</td>
</tr>
<tr>
<td>byte validStation</td>
<td>Value from 0-255 defining to which station this menu item will be visible.</td>
</tr>
<tr>
<td>string taskList</td>
<td>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.</td>
</tr>
<tr>
<td>byte validPosition</td>
<td>Value from 0-255 defining at which robot position this menu item will be visible. Three predefined positions are available: • GAP_SHOW_SAFE • GAP_SHOW_SERVICE • GAP_SHOW_ALWAYS</td>
</tr>
<tr>
<td>bool allowAfterError</td>
<td>True if the menu item should be shown after an error.</td>
</tr>
<tr>
<td>num type</td>
<td>1 = Setup menu, 2 = Service menu.</td>
</tr>
<tr>
<td>byte minUserLevel</td>
<td>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.</td>
</tr>
<tr>
<td>bool blockOtherTasks</td>
<td>If this is set to True all other tasks will be blocked during the execution of this routine.</td>
</tr>
<tr>
<td>num plcCode</td>
<td>Unique identifier index for PLC interfaces. Note: Used only when an external system is controlling the robot system.</td>
</tr>
</tbody>
</table>

### Partdata

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string pathProcName</td>
<td>The procedure that will be called when running this part. The pathProcName 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.</td>
</tr>
<tr>
<td>string description</td>
<td>A short description of the part.</td>
</tr>
<tr>
<td>string taskList</td>
<td>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.</td>
</tr>
<tr>
<td>byte validStation</td>
<td>Value from 0-255 defining for which station this part will be valid.</td>
</tr>
</tbody>
</table>

Continues on next page
### Partdata

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

#### 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.
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
```
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.
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.

Note

Production Screen requires a separate option.

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

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.
Enable/disable

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

- the execution state must be in running mode.
- the menu data 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.
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.

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.
2.3.2 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.

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.
New menudata dialog

<table>
<thead>
<tr>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure name</td>
<td>This is the procedure that will be called when menu is executed. A default name is suggested when creating new menudata. Select the field to change the suggested name by typing a new name in the alpha pad.</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Run in tasks</td>
<td>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.</td>
</tr>
<tr>
<td>Procedure valid</td>
<td>This field cannot be edited. It indicates if the procedure name and menudata instance name is valid in all tasks in the task list.</td>
</tr>
<tr>
<td>Valid on stations</td>
<td>Select the stations this setup or service procedure will be valid on.</td>
</tr>
<tr>
<td>Valid position</td>
<td>Select the position this setup or service procedure will be valid for. Three predefined positions are available.</td>
</tr>
<tr>
<td>Block other tasks</td>
<td>If this is set to True all other tasks will be blocked during the execution of this routine.</td>
</tr>
<tr>
<td>PLC code</td>
<td>Unique identifier index for PLC interfaces.</td>
</tr>
<tr>
<td>User Level</td>
<td>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.</td>
</tr>
<tr>
<td>Menudata instance</td>
<td>The name of the menudata instance in RAPID.</td>
</tr>
<tr>
<td>Declared in module</td>
<td>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 menudata and procedure will be placed in this module. Note: Only normal program modules will be visible in the list.</td>
</tr>
</tbody>
</table>

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.
2 Tap New on the command bar.

3 Select Menu name in list and tap ABC....
4. Enter the path to the module and procedure name separated with @. If module does not exist, it will be created.

5. Or

Select Menu name in list and tap Browse....
6 Tap button ... to browse for module.

7 Select module.
8 Select procedure and tap OK.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Select procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu name</td>
<td>menuData2</td>
<td>DynamicMenuProc</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runs in tasks</td>
<td>T_ROB1</td>
<td></td>
</tr>
<tr>
<td>Procedure(s) valid</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Valid on stations</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Valid position</td>
<td>Show at safe and service</td>
<td></td>
</tr>
</tbody>
</table>

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

2. Select the menu to edit.

3. Tap Edit.

The Edit menu data 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 menu data.
## Edit Setup or Service menu dialog

<table>
<thead>
<tr>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure name</td>
<td>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.</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Run in tasks</td>
<td>These are the tasks this setup or service procedure should be declared in.</td>
</tr>
<tr>
<td>Select the line and check the boxes in the window that appears on the right side.</td>
<td>If more than one task is simultaneously selected it means that these tasks will be executed simultaneously.</td>
</tr>
<tr>
<td>Procedure valid</td>
<td>This field cannot be edited. It indicates if the procedure name and menudata instance name is valid in all tasks in the task list.</td>
</tr>
<tr>
<td>Valid on stations</td>
<td>Select the stations this setup or service procedure will be valid on.</td>
</tr>
<tr>
<td>Valid position</td>
<td>Select the position this setup or service procedure will be valid for. Three predefined positions are available.</td>
</tr>
<tr>
<td>Block other tasks</td>
<td>If this is set to True all other tasks will be blocked during the execution of this routine.</td>
</tr>
<tr>
<td>PLC code</td>
<td>Unique identifier index for PLC interfaces.</td>
</tr>
<tr>
<td>User Level</td>
<td>Value from 0-255 defining the minimum user level required to run this menu, if using UAS. See <a href="#">User Authorization System settings on page 58</a> for more information.</td>
</tr>
<tr>
<td>Menudata instance</td>
<td>The name of the menudata instance in RAPID. This field is not possible to edit.</td>
</tr>
<tr>
<td>Declared in module</td>
<td>The module where the data and the procedure will be declared. Note: This field is not possible to edit.</td>
</tr>
</tbody>
</table>
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".

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

![Image of production information window]

**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/syncronizing for other task in list.
- Graphical view of production cycle
2 Production Manager user interface

2.4 Production Information window

Continued
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.

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
To start debugging:

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

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
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.

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.
### New parts dialog

<table>
<thead>
<tr>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part name</td>
<td>This is the procedure that will be called during production. It typically contains process instructions such as ArcL, SpotL. A default name is suggested when creating new partdata. Select the field to change the suggested name by typing a new name in the alpha pad.</td>
</tr>
<tr>
<td>Description</td>
<td>A custom string that describes the part.</td>
</tr>
<tr>
<td>Run in tasks</td>
<td>These are the tasks this part 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 selected it means that these tasks will be executed simultaneous.</td>
</tr>
<tr>
<td>Procedure valid</td>
<td>This field cannot be edited. It indicates if the Part name and partdata instance name are valid in all tasks in the task list.</td>
</tr>
<tr>
<td>Valid on stations</td>
<td>Select the stations this part will be valid on.</td>
</tr>
<tr>
<td>Partdata instance</td>
<td>The name of the partdata instance in RAPID.</td>
</tr>
<tr>
<td>Declared in module</td>
<td>Select the module where the data and the part procedure will be declared. It is possible to create a new module for the part. 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 partdata and procedure will be placed in this module. Note: Only normal program modules will be visible in the list.</td>
</tr>
</tbody>
</table>
2 Production Manager user interface

2.5.2 Create a new part

Continued

<table>
<thead>
<tr>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced part</td>
<td>Connect an advanced part to this part. See Example 1, advanced part on page 16.</td>
</tr>
</tbody>
</table>

Create a new dynamic part

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

![Create new dynamic part](xx1400002354)

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

![Create new dynamic part](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.

Or

Select Part name in list and tap Browse….

Continues on next page
6 Tap the button ... to browse for module.

7 Select module.
2 Production Manager user interface

2.5.2 Create a new part

Continued

8 Select procedure and tap OK.

![Create new part data](image)

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

![Create new part data](image)
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.

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.
### Edit partdata dialog

<table>
<thead>
<tr>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Part name</strong></td>
<td>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. Use the dropdown list to search for procedures in another module.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A custom string that describes the part.</td>
</tr>
<tr>
<td><strong>Run in tasks</strong></td>
<td>These are the tasks this part 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 selected, it means that these tasks will be executed simultaneous.</td>
</tr>
<tr>
<td><strong>Procedure valid</strong></td>
<td>This field cannot be edited. It indicates if the Part name and partdata instance name are valid in all tasks in the task list.</td>
</tr>
<tr>
<td><strong>Valid on stations</strong></td>
<td>Select the stations this part will be valid on.</td>
</tr>
<tr>
<td><strong>Partdata instance</strong></td>
<td>The name of the partdata instance in RAPID. This field is not possible to edit.</td>
</tr>
<tr>
<td><strong>Declared in module</strong></td>
<td>The module where the data and the part procedure is declared. This field is not possible to edit.</td>
</tr>
<tr>
<td><strong>Advanced part</strong></td>
<td>Connect an advanced part to this part. See Example 1, advanced part on page 16.</td>
</tr>
</tbody>
</table>
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.

Note

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

<table>
<thead>
<tr>
<th>State/Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>No icon when the execution engine is running. This is the normal state.</td>
</tr>
<tr>
<td>Stopped</td>
<td>The task has stopped.</td>
</tr>
<tr>
<td>Busy</td>
<td>The task is producing a part or running a menu.</td>
</tr>
<tr>
<td>Blocked</td>
<td>The task is blocked by another task currently running a Setup or Service menu. See <em>Menu data on page 15</em>.</td>
</tr>
</tbody>
</table>

Location of state icons

The state icons are located at each tab.

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.

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

Production Manager Process Settings

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

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

*Production Manager API State* specifies the signals for tasks state.

<table>
<thead>
<tr>
<th>System parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task name</td>
<td>Name of the task for which the configuration is valid for.</td>
</tr>
<tr>
<td>At-Safe DI</td>
<td>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.</td>
</tr>
<tr>
<td>At-Service DI</td>
<td>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.</td>
</tr>
<tr>
<td>Running out signal</td>
<td>Digital output signal specifying that Prod Mgr task is running.</td>
</tr>
<tr>
<td>Ready out signal</td>
<td>Digital output signal specifying that Prod Mgr task is ready for new order.</td>
</tr>
<tr>
<td>Error group out signal</td>
<td>Group output signal for error codes. If not configured, the error code will be mirrored to the PLC group output signal instead. PLC codes&lt;=99 can be used if configured.</td>
</tr>
<tr>
<td>Error strobe out signal</td>
<td>If defined this signal will go high when error occurs.</td>
</tr>
<tr>
<td>Error ack in signal</td>
<td>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).</td>
</tr>
</tbody>
</table>

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.

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

Continues on next page
### System parameter | Description
---|---
_run ack signal_ | Acknowledge signal used for handshake.
_run ack timeout_ | Defines the timeout when waiting for the run part or run menu signal to go low.
_plc group in signal_ | Group input signal that defines the PLC order.
_allow 0 value for plc_ | Flag for allowing 0 value for PLC. If set, search for `partdata` with value 0 for PLC is done. Only used if PLC group in is configured.
_plc group out signal_ | 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 _Production Manager API State_.
_no reset of plc out signal_ | Flag for specifying if PLC should be reset after part go signal. TRUE means no reset of PLC out signal. Default value is FALSE.
_at station 1 insignal_ | Input signal that specifies that robot/task is at station 1.
_at station 2 insignal_ | Input signal that specifies that robot/task is at station 2.
_at station 3 insignal_ | Input signal that specifies that robot/task is at station 3.
_at station 4 insignal_ | Input signal that specifies that robot/task is at station 4.
_at station 5 insignal_ | Input signal that specifies that robot/task is at station 5.
_at station 6 insignal_ | Input signal that specifies that robot/task is at station 6.
_at station 7 insignal_ | Input signal that specifies that robot/task is at station 7.
_at station 8 insignal_ | Input signal that specifies that robot/task is at station 8.
_station 1 next insignal_ | Input signal that specifies next station 1 for robot/task.
_station 2 next insignal_ | Input signal that specifies next station 2 for robot/task.
_station 3 next insignal_ | Input signal that specifies next station 3 for robot/task.
_station 4 next insignal_ | Input signal that specifies next station 4 for robot/task.
_station 5 next insignal_ | Input signal that specifies next station 5 for robot/task.
_station 6 next insignal_ | Input signal that specifies next station 6 for robot/task.
_station 7 next insignal_ | Input signal that specifies next station 7 for robot/task.
_station 8 next insignal_ | Input signal that specifies next station 8 for robot/task.
_check aborted part_ | Flag for check if part is finished.
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.

```
-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.

<table>
<thead>
<tr>
<th>System parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Task name</em></td>
<td>Name of the task for which the configuration is valid for.</td>
</tr>
<tr>
<td><em>Instance name</em></td>
<td>Name of executing <em>partdata</em> instance.</td>
</tr>
<tr>
<td><em>Station</em></td>
<td>Executing station robot/task.</td>
</tr>
</tbody>
</table>
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>"
```
3.3 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.

<table>
<thead>
<tr>
<th>Application Grant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Parts</td>
<td>If true, the user is allowed to select parts in stations in the Part Handling window. Requires the controller grant Modify current value.</td>
</tr>
<tr>
<td>Edit Parts</td>
<td>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.</td>
</tr>
<tr>
<td>Debug Parts</td>
<td>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.</td>
</tr>
<tr>
<td>Edit Menus</td>
<td>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.</td>
</tr>
<tr>
<td>Run Menu User Level</td>
<td>This grant level is connected to the byte minUserLevel field in the menudata. The logged on user is allowed to run the menu if this grant is true and the minUserLevel field in the menudata &lt;= Run Menu User Level. Min value: 0. Max value: 255. Requires the controller grant Modify current value and I/O write access.</td>
</tr>
</tbody>
</table>

- **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 Seam Displacement.

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.
Now the two different menu data 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.
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`.

<table>
<thead>
<tr>
<th>Task name</th>
<th>Run part in signal</th>
<th>Run menu in signal</th>
<th>Run ack signal</th>
<th>PLC group in signal</th>
<th>PLC group out signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T_002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T_003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T_004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
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

```plaintext
PERS partdata pdProgStn1:="["Program station 1", "T_ROB1:T_ROB2:T_ROB3:T_POS1",1, "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
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];
4 Production Manager PLC support

4.2 How to run Production Manager from PLC via RAPID

Continued

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

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

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

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 Proc1 ()
  taskNr := GAP_TASK_NO;
  PMgrRunMenu(TaskNumber:=taskNr, mnuBE);
ENDPROC
```

Arguments

PMgrRunMenu [\TaskNumber] Menu

<table>
<thead>
<tr>
<th>\TaskNumber</th>
<th>Data type: num</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

Menu

<table>
<thead>
<tr>
<th>Data type: menudata</th>
</tr>
</thead>
<tbody>
<tr>
<td>The menu to execute.</td>
</tr>
</tbody>
</table>

Syntax

PMgrRunMenu

```
[ 'TaskNumber' :=' < expression (IN) of num > ]
[ Menu '=-' ] < var or pers (INOUT) of menudata > ';'
```
5 RAPID references

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 Proc1 ()
    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 Proc1 ()
    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 > ')'
5.2.3 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 Proc1 ()
   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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP_STATE_UNKN</td>
<td>0</td>
<td>Unknown state/not running</td>
</tr>
<tr>
<td>GAP_STATE_IDLE</td>
<td>1</td>
<td>Executing but idle</td>
</tr>
<tr>
<td>GAP_STATE_SETUP</td>
<td>2</td>
<td>Executing setup routine</td>
</tr>
<tr>
<td>GAP_STATE_PART</td>
<td>3</td>
<td>Executing part</td>
</tr>
<tr>
<td>GAP_STATE_SERV</td>
<td>4</td>
<td>Executing service routine</td>
</tr>
</tbody>
</table>

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 Proc1 ()
   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]

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

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

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 Proc1 ()
    taskNr := GAP_TASK_NO;
    nextStation:=PMgrNextStation(TaskNumber:=taskNr);
    TPWrite "Next station is" + ValToStr(nextStation);
ENDPROC

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

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.

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**

```plaintext
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

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

```plaintext
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**

```plaintext
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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP_TASK_NO</td>
<td>The Production Manager specific task index for current task.</td>
</tr>
<tr>
<td>GAP_TASK_NAME</td>
<td>The task name of current task.</td>
</tr>
</tbody>
</table>

Menus

Type field

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP_SETUP_TYPE</td>
<td>1</td>
</tr>
<tr>
<td>GAP_SERVICE_TYPE</td>
<td>2</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP_SHOW_NEVER</td>
<td>0</td>
</tr>
<tr>
<td>GAP_SHOW_SAFE</td>
<td>1</td>
</tr>
<tr>
<td>GAP_SHOW_SERVICE</td>
<td>2</td>
</tr>
<tr>
<td>GAP_SHOW_ALWAYS</td>
<td>255</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP_STATE_UNKN</td>
<td>0</td>
<td>Unknown state/not running</td>
</tr>
<tr>
<td>GAP_STATE_IDLE</td>
<td>1</td>
<td>Executing but idle</td>
</tr>
<tr>
<td>GAP_STATE_SETUP</td>
<td>2</td>
<td>Executing setup routine</td>
</tr>
<tr>
<td>GAP_STATE_PART</td>
<td>3</td>
<td>Executing part</td>
</tr>
</tbody>
</table>

Continues on next page
5 RAPID references

5.3 Public constants

Example:

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP_STATE_SERV</td>
<td>4</td>
<td>Executing service routine</td>
</tr>
</tbody>
</table>

Events

Public constants to be used when defining events:

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

2. In the Production Manager main menu select Part Handling.
3 Tap Seam Displacement on the bottom menu.

![Diagram showing Seam Displacement options]

Selected parts for available stations

<table>
<thead>
<tr>
<th>Station</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Station 1</td>
<td>No part selected</td>
</tr>
<tr>
<td>Station 2</td>
<td>No part selected</td>
</tr>
<tr>
<td>Station 3</td>
<td>No part selected</td>
</tr>
<tr>
<td>Station 4</td>
<td>No part selected</td>
</tr>
<tr>
<td>Station 5</td>
<td>No part selected</td>
</tr>
</tbody>
</table>

Select Part: Seam Displacement

xx1400002381
6 Seam Displacement options

6.3 Functions available in Seam Displacement

FlexPendant screen appearance

Seam Displacement dialog

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target selection</strong></td>
</tr>
<tr>
<td><strong>Note</strong></td>
</tr>
<tr>
<td><strong>Selected targets</strong></td>
</tr>
<tr>
<td><strong>File</strong></td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Tune targets</strong></td>
</tr>
<tr>
<td><strong>APPLY</strong></td>
</tr>
</tbody>
</table>

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
ett 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_SHOWSAFE, 79
GAP SHOW SERVICE, 79
GAP_STATE_IDLE, 79
GAP_STATE_PART, 79
GAP_STATE_SERV, 80
GAP_STATE_UNKN, 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