

---

ROBOTICS

# Application manual

## Dispense



Trace back information:  
Workspace Main version a499  
Checked in 2023-02-10  
Skribenta version 5.5.019

# **Application manual**

## **Dispense**

**RobotWare 6.15**

**Document ID: 3HAC050995-001**

**Revision: E**

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 2004-2023 ABB. All rights reserved.  
Specifications subject to change without notice.

# Table of contents

Overview of this manual .....	7
Product documentation .....	9
<b>1 Introduction to RobotWare Dispense</b> .....	<b>11</b>
1.1 RobotWare Dispense .....	11
1.2 Dispensing features .....	12
1.3 Programming principles .....	13
<b>2 Programming</b> .....	<b>15</b>
2.1 Introduction to programming .....	15
2.2 Programming dispense instructions .....	17
2.3 Editing dispense instructions .....	19
2.4 Testing dispense instructions .....	20
2.5 Programming when short beads are used .....	21
2.6 Softening the robot movements .....	24
2.7 Using the Dispense Restart function .....	25
2.8 Using Dispense in a MultiMove system .....	27
<b>3 RAPID reference</b> .....	<b>29</b>
3.1 Instructions .....	29
3.1.1 DispL/DispC .....	29
3.2 Data types .....	40
3.2.1 beaddata .....	40
3.2.2 equipdata .....	43
<b>4 System modules</b> .....	<b>49</b>
4.1 DPUSER .....	49
4.2 Routines .....	50
<b>5 System parameters</b> .....	<b>51</b>
5.1 Topic I/O .....	51
5.2 Topic Process .....	53
5.2.1 Type Dispense GUI .....	53
5.2.1.1 The Dispense GUI type .....	53
5.2.1.2 Equipment .....	54
5.2.1.3 Equipment visible .....	55
5.2.1.4 Equipment name .....	56
5.2.1.5 Flow1 visible .....	57
5.2.1.6 Flow1 name .....	58
5.2.1.7 Flow2 visible .....	59
5.2.1.8 Flow2 name .....	60
5.2.2 Type Dispense Restart .....	61
5.2.2.1 The Dispense Restart type .....	61
5.2.2.2 Equipment .....	62
5.2.2.3 Restart type .....	63
5.2.2.4 Gun on anticipate .....	64
5.2.2.5 Flow anticipate .....	65
5.2.2.6 Preflow value flow1 .....	66
5.2.2.7 Preflow value flow2 .....	67
5.2.2.8 Backward key .....	68
5.2.2.9 Backward distance after stop .....	69
5.2.2.10 Backward distance after quick stop .....	70
5.2.3 Type Dispense Signals .....	71
5.2.3.1 The Dispense Signals type .....	71
5.2.3.2 Equipment .....	72

## Table of contents

---

5.2.3.3	Flow1 .....	73
5.2.3.4	Flow2 .....	74
5.2.3.5	OnOff .....	75
5.2.3.6	Dispense active .....	76
5.2.3.7	Dispense error .....	77
5.2.3.8	Overspeed .....	78
5.2.3.9	Switch value .....	79
<b>6</b>	<b>FlexPendant Interface</b> .....	<b>81</b>
6.1	Application overview .....	81
6.2	Main View .....	83
6.3	Process Data .....	85
6.3.1	Introduction .....	85
6.3.2	Editing data .....	86
6.4	Process Signals .....	87
<b>7</b>	<b>Customizing RobotWare-Dispense</b> .....	<b>89</b>
7.1	Introduction .....	89
7.2	Files to be changed during the customizing .....	90
7.3	Customizing guides .....	92
<b>8</b>	<b>Dispense options</b> .....	<b>99</b>
8.1	DispensePac Support (901-1) .....	99
8.1.1	Introduction .....	99
8.1.2	RAPID reference for DispensePac Support .....	100
8.1.2.1	SetTmSignal .....	100
8.1.2.2	GetSignal .....	103
<b>Index</b>		<b>105</b>

---

# Overview of this manual

## About this manual

This manual describes the option *Dispense* and contains instructions for the configuration.



### Note

It is the responsibility of the integrator to provide safety and user guides for the robot system.

This manual should be used during installation and configuration of the option *Dispense*.

## Who should read this manual?

This manual is intended for:

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

## Prerequisites

The reader should have the required knowledge of:

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

## References

References	Document ID
<i>Operating manual - IRC5 with FlexPendant</i>	3HAC050941-001
<i>Operating manual - RobotStudio</i>	3HAC032104-001
<i>Technical reference manual - System parameters</i>	3HAC050948-001
<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>	3HAC050917-001
<i>Technical reference manual - RAPID Overview</i>	3HAC050947-001

## Revisions

Revision	Description
-	Released with RobotWare 6.0.
A	Released with RobotWare 6.03. <ul style="list-style-type: none"> <li>• Added the possibility to keep the analog dispense signals between beads, see <a href="#">equipdata on page 43</a> and <a href="#">Keeping the signals between beads on page 22</a>.</li> <li>• Added the argument <i>Trigg Data Array Parameter</i>, see <a href="#">DispL/DispC on page 29</a>.</li> </ul>

Continues on next page

Revision	Description
B	Released with RobotWare 6.07. <ul style="list-style-type: none"><li>• Updated information for the argument D, see <a href="#">DispL/DispC on page 29</a>.</li></ul>
C	Released with RobotWare 6.09. <ul style="list-style-type: none"><li>• Limitation information updated for instructions <code>DispL/DispC</code> in section <a href="#">DispL/DispC on page 29</a>.</li></ul>
D	Released with RobotWare 6.13. <ul style="list-style-type: none"><li>• Added clarification regarding the included signals.</li></ul>
E	Released with RobotWare 6.15. <ul style="list-style-type: none"><li>• Added limitation about system inputs in <a href="#">Programming dispense instructions on page 17</a></li></ul>



---

# Product documentation

---

## Categories for user documentation from ABB Robotics

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



### Tip

All documents can be found via myABB Business Portal, [www.abb.com/myABB](http://www.abb.com/myABB).

---

## Product manuals

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

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

---

## Technical reference manuals

The technical reference manuals describe reference information for robotics products, for example lubrication, the RAPID language, and system parameters.

---

## Application manuals

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

An application manual generally contains information about:

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

*Continues on next page*

- Examples of how to use the application.

---

### Operating manuals

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

# 1 Introduction to RobotWare Dispense

## 1.1 RobotWare Dispense

---

### Introduction

The RobotWare-Dispense package provides support for different types of dispensing processes such as gluing, sealing and similar.

The RobotWare-Dispense application provides fast and accurate positioning combined with a flexible process control. Communication with the dispensing equipment is carried out by digital and analog outputs.

RobotWare-Dispense is a package that can be extensively customized. User data and routines can be adapted to suit a specific dispensing equipment and environmental situation.

# 1 Introduction to RobotWare Dispense

---

## 1.2 Dispensing features

### 1.2 Dispensing features

---

#### Option content

- Dispensing instructions for both linear and circular paths.
- Fast and accurate positioning and process control.
- Handling of on/off guns as well as proportional guns.
- Speed proportional or constant analog outputs.
- Four different gun equipment can be handled in the same program, each controlled by 1-5 digital output signals and/or 1-2 analog output signals.
- Possibility to use different anticipated times for the digital and analog signals.
- Equipment delay compensation for the TCP speed proportional analog signals.
- Dispensing in wet or dry mode.
- Possibility to restart an interrupted dispense sequence.
- Possibility to automatically reduce the robot acceleration/deceleration during dispensing.
- Wide customizing possibilities.
- A dedicated Dispense operator interface on the FlexPendant.
- RobotWare-Dispense can also be used in MultiMove systems.

## 1.3 Programming principles

### Introduction

Both the robot movement and the dispensing process control are embedded in the instructions, `DispL` and `DispC`. See [RAPID reference on page 29](#)

The dispensing process is specified by:

- Bead specific dispensing data. See [beaddata on page 40](#).
- Equipment specific dispensing data. See [equipdata on page 43](#).
- RAPID routines and global data for customizing purposes. See [System modules on page 49](#).
- System parameters. See [System parameters on page 51](#).

### Dispensing instructions

Instruction	Used to:
<code>DispL</code>	Move the TCP along a linear path and perform dispensing with the given data.
<code>DispC</code>	Move the TCP along a circular path and perform dispensing with the given data.

### Dispensing data

Data type	Used to define:
<code>beaddata</code>	Dispensing data for the different beads.
<code>equipdata</code>	Equipment specific dispensing data.

### Signals

The available signals are set during bead segments according to the configuration possibilities described in this manual, and in relation to the position of the tool center point and the programmed targets. The signals can be used for any equipment-specific purposes, and many setups will not have use for all signals. Some equipment might use multiple signals for controlling two physical pieces of equipment at once, and others may use multiple signals for more advanced equipment functions.

The digital gun signals can typically be used for simple purposes requiring only an on/off state during a bead, such as opening a gun or triggering an external event.

The analog flow signals can typically be used for purposes where different levels are required, such as pressure or function speed. These levels can be adjusted according to bead type and can also optionally be automatically adjusted during robot movement in relation to the actual speed of the tool center point.

**This page is intentionally left blank**

## 2 Programming

### 2.1 Introduction to programming

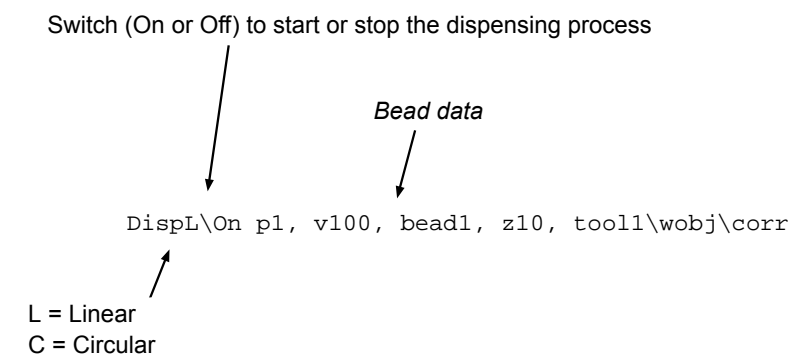
#### Introduction

When RobotWare-Dispense option is added to the RobotWare base system, it is ready to use with a default functionality. However, the functionality can be customized to adapt the package to a specific dispensing equipment, see [Customizing RobotWare-Dispense on page 89](#).

This chapter describes primarily the default setup.

#### Dispense instructions

A dispense instruction contains the same type of information as a Move instruction. However, each dispense instruction also include further arguments for the dispensing process. See [DispL/DispC on page 29](#).



xx1200000143

#### Dispense data types

The dispense data should be defined before programming. The dispense data is divided in two types:

- equipdata
- beaddata

#### equipdata

The data type `equipdata` is used to hold equipment specific data. This is dispensing data which normally does not vary between different beads. It is possible to use up to four different pieces of equipment. `equipdata` for equipment 1-4 are located in an array (`equipd`) in the system module DPUSER. The data in current `equipdata` influences the process when dispensing instructions are executed.

`equipdata` has the following components:

- Information string (80 characters)
- Reference speed for the scale calculations.
- Acceleration/deceleration limits for the robot movements.

*Continues on next page*

## 2 Programming

---

### 2.1 Introduction to programming

*Continued*

- Predict times for the digital (gun on and off) signals.
- Predict times for the analog flow signals.
- Equipment delay compensation times.
- Flow rate correction factors.

Change the data in the *equipd* array so it corresponds to the equipment used. The data is correctly calibrated when the bead is started/changed/ended in the programmed positions and when the bead width is constant in corners. See [equipdata on page 43](#).

#### beaddata

*beaddata* is used to hold bead specific data. This is the dispensing data that determines the appearance of the bead and which normally varies between different beads. One *beaddata* (*bead1*) is predefined as default in the dispensing system module DPUSER.

*beaddata* has the following components:

- Information string (80 characters)
- Flow rates for flow1 and flow2
- Flow types for flow1 and flow2
- Equipment number
- Gun number

We recommend that one *beaddata* for each used bead type is defined and stored in DPUSER, before the instruction sequence is programmed. See [beaddata on page 40](#).

---

#### Dispense programming philosophy

RobotWare-Dispense supports a bead oriented programming philosophy.

Parameters affecting the characteristics of the bead are included in the *beaddata* for each beadtype. This technique separates the defining of the bead parameters from robot path teaching and simplifies off line programming as well as manual teaching and touch up.

In the default *beaddata* and *equipdata* version the most used data components are present. During customizing it is possible to hide (delete) components if they not are valid for the specific process or equipment. it is also possible to add some components if desired. For more information, see [Customizing RobotWare-Dispense on page 89](#).



## 2.2 Programming dispense instructions

### Programming

	Action	Note
1	Jog the robot to the desired position for the dispensing start.	
2	Tap <b>Add Instruction</b> and select the <b>Dispense</b> pick list.	
3	Add the instruction <code>DispL\On</code> or <code>DispC\On</code> .	The arguments are set in relation to the latest programmed dispense instruction.
4	If needed, select <code>beaddata</code> and change other arguments.	
5	Jog to next position on the bead.	Normally a point where the path is changed or a position where the bead size is changed.
6	Add another instruction, <code>DispL</code> or <code>DispC</code> .	
7	If needed, change the arguments.	
8	Continue this way until the last position on the bead is reached.	
9	Add the end instruction <code>DispL\Off</code> or <code>DispC\Off</code> , and change the arguments if needed.	

### Limitations

One equipment is controlled by 1-5 gun on/off signals and 1-2 analog flow signals. It is possible to use up to four different pieces of equipment.

It is not possible to use different pieces of equipment within one sequence of dispensing instructions (from `DispL\On` to `DispL\Off`).

It is not possible to change from flow control type 2 to flow control type 1 within one sequence of dispensing instructions.

It is not recommended to use system inputs to change the programmed TCP speed during dispensing. This includes the following features:

- System input *Limit Speed*
- Setting *Speed Override* from the FlexPendant
- System input *Set Speed Override*

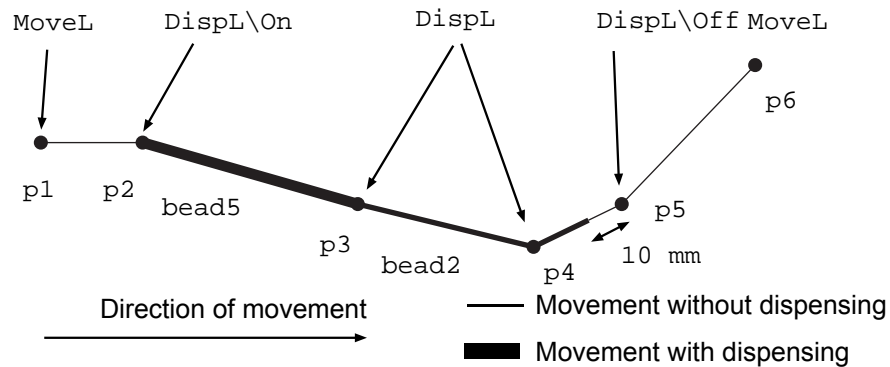
*Continues on next page*

## 2 Programming

### 2.2 Programming dispense instructions

Continued

#### Programming example



xx1200000606

```
bead5
  info = "width 5mm"
  flow1 = 80
  flow2 = 40
  flow1_type = 2
  flow2_type = 1
  equip_no = 1
  gun_no = 1

bead2
  info = "width 2mm"
  flow1 = 35
  flow2 = 20
  flow1_type = 2
  flow2_type = 1
  equip_no = 1
  gun_no = 1

equipd{1}
  info = "equipment 1"
  on_time = 0.05
  off_time = 0.05
  ref_speed = 800
  acc_max = 5
  decel_max = 5

  f11_on_time = 0.10
  f11_off_time = 0
  f11_inc_time = 0.06
  f11_dec_time = 0.12
  f11_delay = 0.05
  f11_corr = 100

  f12_on_time = 0.15
  f12_off_time = 0
  f12_inc_time = 0.08
  f12_dec_time = 0.1
  f12_delay = 0.04
  f12_corr = 100
```

#### RAPID code sequence:

```
MoveL p1, v600, fine, tool1;
DispL \On, p2, v400, bead5, z10, tool1;
DispL p3, v400, bead2, z10, tool1;
DispL p4, v500, bead2, z10, tool1;
DispL \Off, p5, v500, bead2 \D:=10, z10, tool1;
MoveL p6, v600, fine, tool1;
```

## 2.3 Editing dispense instructions

### Changing a beaddata parameter

	Action
1	Select <b>Data</b> in the Dispense GUI.
2	Select <code>beaddata</code> in the <b>Data</b> menu in the command bar.
3	Select <b>desired</b> <code>beaddata</code> and change values.
4	Tap <b>OK</b> .

### Changing to another beaddata

	Action
1	Select <b>current</b> <code>beaddata</code> in the instruction.
2	Tap <b>Edit</b> and then <b>Change Selected</b> .
3	Select <code>beaddata</code> from the list.
4	Tap <b>OK</b> .

## 2 Programming

---

### 2.4 Testing dispense instructions

#### 2.4 Testing dispense instructions

---

##### Testing without dispensing

It is possible to run the program in simulation mode without activating any dispense signals. This can be done by setting the DPUSER data *dp\_dry* to TRUE.

This can also be done by tapping a button in the **Test Modes** subview in the Dispense GUI, see [FlexPendant Interface on page 81](#).

---

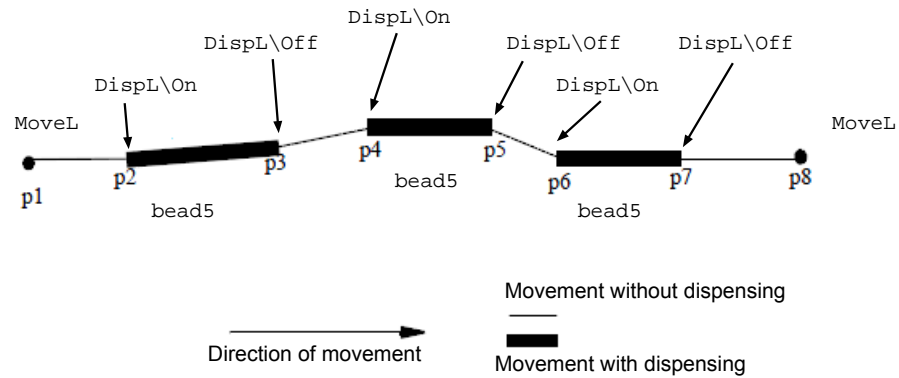
##### Testing dispense instructions step by step

The dispense signals are not activated when dispense instructions are executed step by step.

## 2.5 Programming when short beads are used

### Example 1

In this example a number of short beads are programmed. The On and Off argument is used for all beads that will terminate the dispensing process between the beads.



xx120000627

```

bead5
  info = "width 5mm"
  flow1 = 80
  flow2 = 40
  flow1_type = 2
  flow2_type = 2
  equip_no = 1
  gun_no = 1

```

#### RAPID code sequence:

```

MoveL p1, v600, fine, tool1;
DispL \On, p2, v600, bead5, z50, tool1;
DispL \Off, p3, v600, bead5, z50, tool1;
DispL \On, p4, v600, bead5, z50, tool1;
DispL \Off, p5, v600, bead5, z50, tool1;
DispL \On, p6, v600, bead5, z50, tool1;
DispL \Off, p7, v600, bead5, z50, tool1;
MoveL p8, v600, fine, tool1;

```

In this example the analog flow signals are deactivated between every bead, but often it is desired to run short bead interrupts without deactivating the analog flow signals. Only the digital gun signal(s) are deactivated during the interrupt. This can be done as shown in next example.

*Continues on next page*

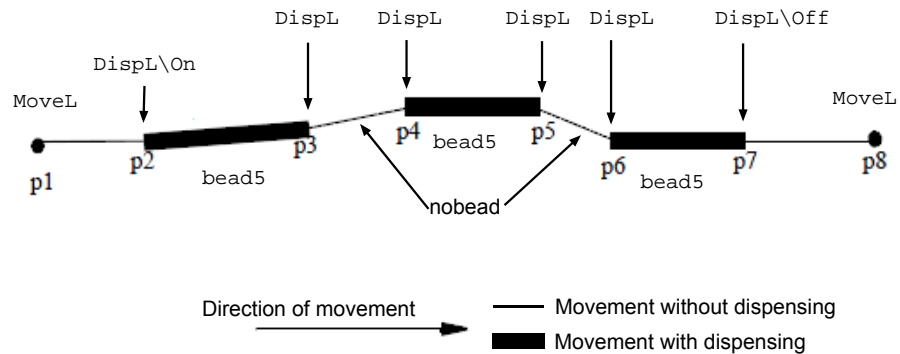
## 2 Programming

### 2.5 Programming when short beads are used

Continued

#### Example 2

The same beads as in previous example are programmed but in this case we do not deactivate the analog flow signals between the beads. A separate bead data, *nobead*, is created to be used between the beads. In this *beaddata* the data component *gun\_no* is set to zero, all other components are set as in previous *beaddata*. The process is started with the *On* argument in the beginning of the first bead, and terminated with the *Off* argument in the end of the last bead.



xx120000627

<pre>bead5   info = "width 5mm"   flow1 = 80   flow2 = 40   flow1_type = 2   flow2_type = 2   equip_no = 1   gun_no = 1</pre>	<pre>nobead   info = "no bead"   flow1 = 80   flow2 = 40   flow1_type = 2   flow2_type = 2   equip_no = 1   gun_no = 0</pre>
---	--

#### RAPID code sequence:

```
MoveL p1, v600, fine, tool1;
DispL \On, p2, v600, bead5, z50, tool1;
DispL p3, v600, nobead, z50, tool1;
DispL p4, v600, bead5, z50, tool1;
DispL p5, v600, nobead, z50, tool1;
DispL p6, v600, bead5, z50, tool1;
DispL \Off, p7, v600, bead5, z50, tool1;
MoveL p8, v600, fine, tool1;
```

#### Keeping the signals between beads

In RobotWare 6.02 a new method is introduced to hold the analog values between beads. This method gives the possibility to decide, for each equipment, if the analog signals shall be reset between beads or not.

If the argument *fl1\_off\_time* or *fl2\_off\_time* is set to -1 then the automatic reset of the flow signal is deactivated. The signal holds the latest value given from the *beaddata* in the *\Off* instruction until start of next bead, see [equipdata on page 43](#).

Continues on next page

By using this method it is possible to program as in example 1, without activating the analog signals between the beads.

---

#### **Limitations**

When short beads or bead interrupts are used in combination with high process speeds it is not possible to use shorter beads than the difference between the on and off time components in equipdata. If the beads are shorter the signals are deactivated before they are activated, that is the beads are not dispensed. In these situations the process speed has to be reduced.

## 2 Programming

---

### 2.6 Softening the robot movements

### 2.6 Softening the robot movements

---

#### Description

If necessary it is possible to reduce the robot acceleration or deceleration to get the best behavior from the used dispensing equipment.

If the data components `acc_max` or `decel_max` are activated in the `equipdata` for a specific equipment, then the limitation is automatically activated in the beginning of the bead and reset to previous value in the end of the bead, where this equipment is used. See [equipdata on page 43](#).



## 2.7 Using the Dispense Restart function

### About the Dispense Restart function

The system parameters for the Dispense Restart function is found in topic *Process*, type *Dispense Restart*.

There is one set of parameters for each available equipment, *Equipment\_1* to *Equipment\_4*.

### Activating the Dispense Restart function

Set the restart type by changing the system parameter *Restart type*. It is possible to set three different restart levels for each equipment.

0	Restart is disconnected. (Default). The process is started in the next bead, i.e. in the next <code>Disp\On</code> instruction.
1	Restart of current bead with possibilities to select wet or dry.
2	Restart of current bead in a no query mode (without questions)

See [DispL/DispC on page 29](#).

### Set up desired restart behavior

Test a common restart situation for each equipment by adjusting the following system parameters to get the desired restart behavior:

Gun on anticipate	The preaction time for the digital gun signal
Flow anticipate	The preflow time for the analog signals
Preflow value flow1	The preflow value for the analog signal 1
Preflow value flow2	The preflow value for the analog signal 2
Backward distance after stop	The backward distance before restart after a program stop
Backward distance after quick stop	The backward distance before restart after a quick stop

For more information about system parameters, see [Topic Process on page 53](#).

### Using a programmable key for the backward on path function

Normally the TCP is moved a predefined distance backward on the path before the process is restarted. But if restart type 1 is selected it is also possible to use one of the FlexPendant programmable keys for activation of the movement. The backward movement is going on as long as the key is pressed.

Activate this function by changing the system parameter *Backward key* for each equipment.

0	No key is used. (Default)
1 - 4	Desired key number

If this function is activated the selected key has to be configured as a programmable key, see [Configuring a key for the backward on path function on page 26](#).

*Continues on next page*

## 2 Programming

---

### 2.7 Using the Dispense Restart function

*Continued*

---

#### Configuring a key for the backward on path function

- 1 On the FlexPendant, open the **Control Panel**.
- 2 Tap **ProgKeys**.
- 3 Select the key to be used.
- 4 Set following parameters for the key:
  - **Type: Output**
  - **Digital output: doBwdOnPath**
  - **Key Pressed: Press/Release**
  - **Allow in Auto: Yes**

## 2.8 Using Dispense in a MultiMove system

### Description

- It is possible to use up to four dispense robots in a MultiMove system.
- Dispense robots can be combined with other process robots in the same MultiMove system.
- It is possible to run dispense instructions independent or synchronized. The optional argument \ID has to be used for synchronized movements.
- In a MultiMove system, as in a single system, up to four different pieces of equipment can be used. In a MultiMove system these equipment can be shared among the different dispense robots. The only restriction is that it is not possible to control same equipment from several robots at the same time.

**This page is intentionally left blank**

## 3 RAPID reference

### 3.1 Instructions

#### 3.1.1 DispL/DispC

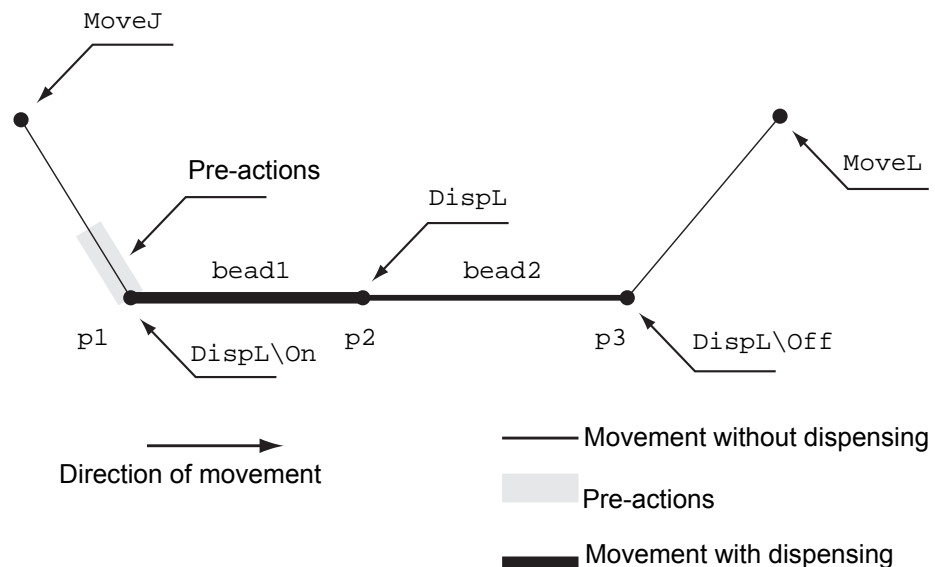
##### Usage

`DispL/DispC` is used in dispensing applications to control the robot motion, gun opening, and the dispensing process. `DispL` moves the TCP linearly to the end position. `DispC` moves the TCP circularly to the end position.

##### Basic examples

In this example a bead starts at point p1, changes to another bead at p2, and then ends at point p3.

```
DispL \On, p1, v250, bead1, z30, tool7;
DispL p2, v250, bead2, z30, tool7;
DispL \Off, p3, v250, bead2, z30, tool7;
```



xx120000146

- 1 The TCP for tool 7 is moved on a linear path to the position p1 with the speed given in v250. Due to the `\On` argument the gun opens and the dispense flow is started in advance on its way to p1. The flow rates and the times when the signals are activated, are specified in `beaddata bead1` and in current `equipdata` in the `equipdata` array in `DPUSER`. See [System modules on page 49](#).
- 2 The TCP is then moved from p1 towards p2 with the flow values given by `beaddata bead1` activated in the preceding dispensing instruction. Before

*Continues on next page*

## 3 RAPID reference

---

### 3.1.1 DispL/DispC

*Continued*

- p2 is reached, the flow values are changed according to the data specified in `bead2`. The time when this is performed is specified in current `equipdata`.
- 3 The TCP is then moved from p2 towards p3 with the flow values specified by `beaddata bead2` activated in the preceding dispensing instruction. Due to the `Off` argument the outputs will be reset, according to the times specified in current `equipdata` connected to `bead2`, before p3 is reached.

---

#### Arguments

```
DispL [\On][\Off] ToPoint[\ID]Speed Bead [\D] Zone Tool [\WObj]
      [\Corr] [\T1] | [\TArray{*}] [\TLoad]
DispC [\On][\Off] CirPoint ToPoint[\ID] Speed Bead [\D] Zone Tool
      [\WObj] [\Corr] [\T1] | [\TArray{*}] [\TLoad]
```

`[\On]`

Data type: `switch`

The argument `\On` is used in the first dispensing instruction in a sequence, to start the dispensing process. The argument may only be used in the first dispense instruction in a sequence, to perform the necessary gun opening and setting of the flow in advance. Executing two consecutive instructions with `\On` argument will result in an error message.

`[\Off]`

Data type: `switch`

The argument `\Off` is used in the last dispense instruction in a sequence, to terminate the process when the programmed position is reached. On the way to the end position the output for opening the gun as well as the flow outputs will be reset according to the given time within the specified data.

`CirPoint`

Data type: `robtarget`

This argument is only used in `DispC`.

The circle point of the robot. The circle point is a point on the circle between the start point and the destination point. To obtain the best accuracy it should be placed about halfway between the start and destination points. If it is placed too close to the start or end point, the robot may give a warning. The circle point is defined as a named position or stored directly in the instruction (if marked with an `*` in the instruction).

`ToPoint`

Data type: `robtarget`

The destination point of the robot and the external axes. It is defined as a named position or stored directly in the instruction (if marked with an `*` in the instruction).

`[\ID]`

*Synchronization id*

Data type: `identno`

The argument `[\ID]` is mandatory in MultiMove systems, if the movement is synchronized or coordinated synchronized. This argument is not allowed in any

*Continues on next page*

other case. The specified id number must be the same in all the cooperating program tasks. By using the id number the movements are not mixed up at the runtime.

#### Speed

**Data type:** speeddata

The speed data that applies to movements. Speed data defines the velocity for the tool center point, the tool reorientation, and external axes.

#### Bead

**Data type:** beaddata

Bead specific data, for example the flow rates and flow types. For further details, see [beaddata on page 40](#).

#### [D]

##### *Distance*

**Data type:** num

The optional argument \D gives the possibility to offset all pre-actions a given distance (mm) on the programmed path.

A positive value will offset the actions to before the programmed position, while a negative value will offset the actions to after the programmed position.

This argument is useful for adjusting the start and stop of individual bead segments without altering the programmed path.

**Limitation:** If the given distance causes an overlap between the pre-actions of this position and the next/previous one, the resultant behavior might not be what is intended. Take care when using this argument in combination with shorter segments.

#### Zone

**Data type:** zonedata

Zone data for the movement. Zone data describes the size of the generated corner path.

#### Tool

**Data type:** tooldata

The tool in use when the robot moves. The tool center point is the point moved to the specified destination position.

#### [WObj]

##### *Work Object*

**Data type:** wobjdata

The work object (coordinate system) to which the robot position in the instruction is related. This argument can be omitted, and if it is, the position is related to the world coordinate system. If, on the other hand, a stationary tool or coordinated external axes are used, this argument must be specified in order to perform a linear movement relative to the work object.

*Continues on next page*

## 3 RAPID reference

---

### 3.1.1 DispL/DispC

*Continued*

[Corr]

#### *Correction*

Data type: `switch`

Correction data written to a corrections entry by the instruction `CorrWrite` will be added to the path and destination position, if this argument is present.

[T1]

#### *Trigg 1*

Data type: `triggdata`

Variable that refers to trigger conditions and trigger activity, defined earlier in the program using the instructions `TriggIO`, `TriggEquip`, `TriggInt`, and similar. Compare with the `TriggL` instruction.

[TArray{\*}]

#### *Trigg Data Array Parameter*

Data type: `triggdata`

Array variable that refers to trigger conditions and trigger activity defined earlier in the program using the instructions `TriggIO`, `TriggEquip`, `TriggInt`, and similar.

The limitation is 10 elements in the array.

[TLoad]

#### *Total Load*

Data type: `loaddata`

The total load used in the movement. The total load is the tool load together with the payload the tool is carrying. If the `\TLoad` argument is used, then the `loaddata` in current `tooldata` is not considered. If the `\TLoad` argument is set to `load0`, then the `\TLoad` argument is not considered and the `loaddata` in current `tooldata` is used instead. For a complete description of the `TLoad` argument, see *MoveL - Moves the robot linearly*.

---

## Program execution

Internal sequence when a `DispL/DispC` instruction is executed:

- The gun starts to move towards the position.
- If the argument `\On` is used, the gun opening output (or outputs) and the analog outputs are set at the specified time before the position is reached. See [equipdata on page 43](#). The Equipment Active signal (for example the output signal `doEqu1Active` for equipment 1) is set together with the gun opening output.
- If nor `\On` or `\Off` is used the gun opening output (or outputs) and the analog outputs are set (or changed) at the specified time before the position is reached.
- If the argument `\Off` is used the gun opening output (or outputs) and the analog outputs are reset at the specified times before the position is reached. The Equipment Active signal is reset together with the digital gun signals.

*Continues on next page*



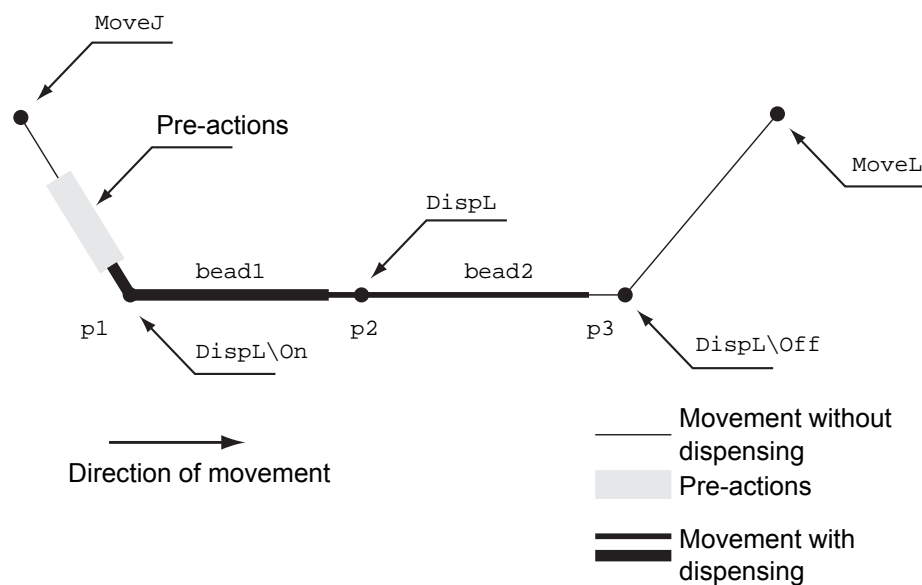
- If the argument `\D` is used, all pre-actions are performed specified times (in equipdata) plus given distance (mm) in advance of the programmed position.
- When the programmed position is reached, the program execution continues with the next instruction.

### More examples

#### Example 1

In this example, everything is the same as in the previous example except that all bead changes are performed an additional distance (50 mm) before the programmed positions are reached.

```
DispL \On, p1, v250, bead1\D:=50, z30, tool7;
DispL p2, v250, bead2 \D:=50, z30, tool7;
DispL \Off, p3, v250, bead2 \D:=50, z30, tool7;
```



xx120000147

#### Example 2

In this example, `DispC` is used:

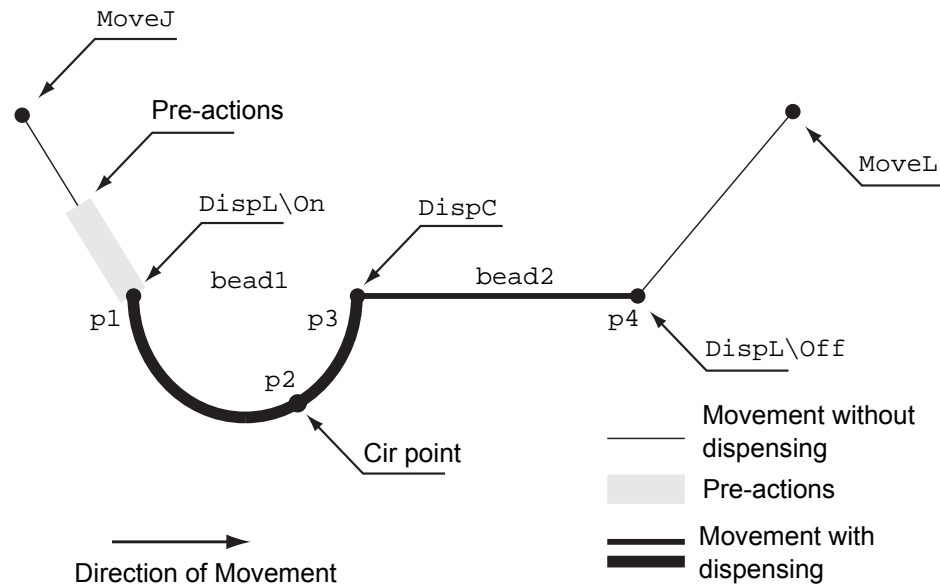
```
DispL \On, p1, v250, bead1, z30, tool7;
DispC p2, p3, v250, bead2, z30, tool7;
DispL \Off, p4, v250, bead2, z30, tool7;
```

Continues on next page

## 3 RAPID reference

### 3.1.1 DispL/DispC

Continued



---

#### Instruction by instruction execution forwards

The gun is closed and motion without dispensing is done.

---

#### Instruction by instruction execution backwards

The gun is closed and the motion is performed backwards without dispensing.

---

#### Simulated dispensing

Activated by setting the variable `dp_dry` to `TRUE`. This will inhibit the gun opening and the flow signals. See [System modules on page 49](#).

---

#### Limitations

`DispL/DispC` cannot be executed in a RAPID routine connected to any of the following special system events: PowerOn, Stop, QStop, Restart, Reset, or Step.

---

#### Error handling

The following error situations are handled:

- Instruction argument error.
- Not permitted data values.
- Start without `\On` argument.
- Start with two instructions with `\On` argument.
- One sequence of Disp instructions cannot use different equipment.
- Stop during execution of dispense instructions.

Error actions:

- The digital on/off signals are cleared.
- The digital *Equipment Active* signal is cleared.

Continues on next page

- The analog flow signals are cleared.
- The digital *Equipment Error* signal is set.
- The program execution is interrupted with the program pointer on the interrupted instruction.

Internal errors are handled internally, but to set the program pointer on the interrupted `Disp` instruction the error is also raised to the user level. But since this internal errors not are handled by the error handlers on the user level, the system error handler automatically will report the error and stop the program execution.

### Flow Control Types

There are two different Flow Control Types for the analog signals. See [beaddata on page 40](#):

Speed independent (1)	The analog flow is a constant value, proportional to the flow rate specified in the current bead data.
TCP speed proportional(2)	The analog flow is directly proportional to the actual robot velocity. When for example the speed is reduced in a corner path, the flow will also be reduced correspondingly.

### Calculation of flow values

The following data is used when the logical flow1 value is calculated. (The logical flow2 is calculated in same way):

flow1	flow rate from current bead data. See <a href="#">bead-data on page 40</a> .	
dp_fl1_corr	global flow rate correction factor. See <a href="#">System modules on page 49</a> .	Default: 100
fl1_corr	flow rate correction factor. See <a href="#">equipdata on page 43</a> .	Default: 100
current speed	current robot speed (mm/s).	
ref_speed	reference speed (mm/s). See <a href="#">equipdata on page 43</a> .	

Calculation of logical flow1 when flow1\_type = Speed independent (1):

$$\text{logical flow1} = \text{flow1} * \text{dp\_fl1\_corr} * \text{fl1\_corr} / 10000$$

Calculation of logical flow1 when flow1\_type = TCP speed proportional (2):

$$\text{logical flow1} = (\text{flow1} * \text{dp\_fl1\_corr} * \text{fl1\_corr} / 10000) * \text{current speed} / \text{ref\_speed}$$

The physical values of the signals are then determined by how the analog signals are configured in the system parameters (relationship between physical and logical values).

For example:

With the default values above and with the default setup for logical max. and min. for the analog outputs (see [System parameters on page 51](#)), the following result is obtained:

If *flow1\_type* = Speed independent (1):

Physical max. value is activated if flow1 = 100 in current bead data.

Or

Continues on next page

## 3 RAPID reference

---

### 3.1.1 DispL/DispC

Continued

If *flow1\_type* = TCP speed proportional (2):

Physical max. value is activated if *flow1* = 100 in current bead data and the actual speed is the same as *ref\_speed* in current *equipdata*.

---

#### Soften the robot movements

If necessary it is possible to reduce the robot acceleration or deceleration to get the best behavior from the used dispensing equipment.

If the data components *acc\_max* or *decel\_max* are activated in the *equipdata* for a specific equipment, then the limitation is automatically activated in the beginning of the bead and reset to previous value in the end of the bead, where this equipment is used. See [equipdata on page 43](#).

---

#### Stop and restart during the dispensing process

It is possible to set three different restart levels for each equipment.

0	Restart is disconnected. (Default). The process is started in the next bead, i.e. in the next <code>Disp\On</code> instruction.
1	Restart of current bead with possibilities to select wet or dry.
2	Restart of current bead in a no query mode (without questions)

For more information, see [Using the Dispense Restart function on page 25](#)

#### Stop sequences

The stop sequence when a `Program Stop` occur during the dispensing process:

	Action
1	The dispensing process is active during the stop phase. If TCP speed proportional analog signals are used the analog flow values will be automatically reduced during the deceleration.
2	The robot is halted on the path.
3	The digital gun signals (for gun on/off) are set to zero.
4	An information text is printed out on the FlexPendant telling if it is possible to restart the process or not.
5	If it not is possible to restart the process the digital Equipment Error signal is activated.

The stop sequence when a quick stop (for example `Emergency Stop`) occur during the dispensing process:

	Action
1	The robot is halted as soon as possible, probably with a deviation from the programmed path.
2	The gun signals are set to zero as soon as possible during the deceleration.
3	An information text is printed out on the FlexPendant telling if it is possible to restart the process or not.
4	If it is not possible to restart the process the digital Equipment Error signal is activated.

Continues on next page

## Restart sequences

The Restart sequence with Restart type 0 (Default):

	Action
1	A regain motion back to the interrupted position on the path is done (after quick stop).
2	The remaining instructions in the current set of dispense instructions are performed as normal positioning instructions. The dispensing is restarted in the next <code>Disp</code> instruction with an <code>\On</code> argument.

The Restart sequence with Restart type 1:

	Action
1	A regain motion back to the interrupted position on the path is done (after quick stop).
2	If it is possible to restart, a text is presented on the display asking if the restart shall be <b>Wet</b> or <b>Dry</b> . It is also possible to select <b>Bwd</b> . <ul style="list-style-type: none"> <li>• <b>Dry</b> means restarting without activating the digital gun signals. If <b>Dry</b> is selected the process is restarted in next <code>Disp\On</code> instruction.</li> <li>• If <b>Wet</b> is pressed the process is restarted directly from current position.</li> <li>• If <b>BWD</b> is selected a new dialog is activated with possibility to move the robot backward on the path. This function is useful if the bead is interrupted before the stop position. When the backward movement is ready previous dialog appear again. (<b>Wet</b> or <b>Dry</b>).</li> </ul>
3	When <b>Wet</b> is selected the digital gun signals and the analog flow signals are activated user defined times before the robot motion starts. ( <code>restart_on_time</code> and <code>restart_fl_time</code> ).
4	Before the signals are activated the user routine <code>dp_restart_proc</code> (in <code>DPUSER</code> ) is executed with possibilities to add some user actions in the restart sequence.

The Restart sequence with Restart type 2:

	Action
1	A regain motion back to the interrupted position on the path is done (after a quick stop).
2	If it is possible to restart, the robot is automatically moved backward on the path a user defined distance. Different distances after program stop and quick stop can be used ( <code>stop_bwddist</code> or <code>qstop_bwddist</code> ).
3	The digital gun signals and the analog flow signals are activated user defined times before the robot motion starts. ( <code>restart_on_time</code> and <code>restart_fl_time</code> ).
4	Before the signals are activated the user routine <code>dp_restart_proc</code> (in <code>DPUSER</code> ) is executed with possibilities to add some user actions in the restart sequence.

**Note**

The bead quality when the process is restarted after a quick stop will not be the same as after a program stop, mainly because that the robot deviates from the programmed path during the deceleration and also because it during emergency stop not is possible to compensate for delays in the dispensing equipment. This sometimes results in too much material in the stop position. However, to avoid these problems safeguarded stops can be configured to be soft which gives a smooth stop on the path in these cases.

Continues on next page

## 3 RAPID reference

### 3.1.1 DispL/DispC

Continued



#### Note

The possibility to move the robot backward on the path is limited to one or a few segments. If it is not possible to move the robot a desired distance backward on the path an error text will appear.



#### Note

It is not possible to restart current bead if Power Failure occur during the dispensing process. After Power On an error text is written on the display. If the program is restarted the process is restarted in next Disp\On instruction.

### Syntax

```
DispL
[[ '\On ', ' ] | [ '\Off ', ' ] ]
[ToPoint := ] <expression (IN) of robtarg>
[ '\ID := ] <expression (IN) of identno> ', '
[Speed := ] <expression (IN) of speeddata> ', '
[Bead := ] <persistent (PERS) of beaddata>
[ '\D := ] <expression (IN) of num> ]
[Zone := ] <expression (IN) of zonedata> ', '
[Tool := ] <persistent (PERS) of tooldata>
[ '\WObj := ] <persistent (PERS) of wobjdata>
[ '\Corr ]
[ '\T1 := ] <variable (VAR) of trigdata> |
[ '\TArray := ] <array variable {*} (VAR) of trigdata> ]
[ '\TLoad := ] <persistent (PERS) of loaddata> ]';

DispC
[[ '\On ', ' ] | [ '\Off ', ' ] ]
[CirPoint := ] <expression (IN) of robtarg> ', '
[ToPoint := ] <expression (IN) of robtarg>
[ '\ID := ] <expression (IN) of identno> ', '
[Speed := ] <expression (IN) of speeddata> ', '
[Bead := ] <persistent (PERS) of beaddata>
[ '\D := ] <expression (IN) of num > ]
[Zone := ] <expression (IN) of zonedata> ', '
[Tool := ] <persistent (PERS) of tooldata>
[ '\WObj := ] <persistent (PERS) of wobjdata>
[ '\Corr ]
[ '\T1 := ] <variable (VAR) of trigdata > |
[ '\TArray := ] <array variable {*} (VAR) of trigdata> ]
[ '\TLoad := ] <persistent (PERS) of loaddata> ]';
```

### Related information

	Described in:
Definition of velocity, speeddata	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>
Definition of zone data, zonedata	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>

Continues on next page

	Described in:
Definition of tool, <code>tooldata</code>	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>
Definition of work objects, <code>wobjdata</code>	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>
Definition of <code>beaddata</code>	<a href="#">beaddata on page 40</a>
Definition of <code>equipdata</code>	<a href="#">equipdata on page 43</a>
Customizing the functionality	<a href="#">Customizing RobotWare-Dispense on page 89</a>
System module DPUSER	<a href="#">System modules on page 49</a>
I/O configuration	<a href="#">Topic I/O on page 51</a>
MoveL	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>
Definition of load, <code>loaddata</code>	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>
Motion in general	<i>Technical reference manual - RAPID Overview</i>
Other positioning instructions	<i>Technical reference manual - RAPID Overview</i>

## 3 RAPID reference

---

### 3.2.1 beaddata

## 3.2 Data types

### 3.2.1 beaddata

---

#### Usage

`beaddata` is used in dispensing applications to hold bead specific data. This is the dispensing data which determines the appearance of the bead and which normally varies between different beads.

---

#### Description

`beaddata` is used in `DispL` and `DispC` instructions and has the following default contents:

- Information string (80 char).
  - Flow rates for flow1 and flow2.
  - Flow control types for flow1 and flow2.
  - Equipment number.
  - Gun number (if multiple guns are used).
- 

#### Arguments

##### Info

*data information*

Data type: `string`

String for external use, maximum 80 characters.

##### flow1

*flow rate*

Data type: `num`

Flow rate for the flow1 signal. Normal values: 0-100

##### flow2

*flow rate*

Data type: `num`

Flow rate for the flow2 signal. Normal values: 0-100

##### fl1\_type

*type of flow1*

Data type: `num`

Flow control type for flow1. Permitted values:

- 0 - Flow1 signal not used
- 1 - Constant, speed independent
- 2 - Speed proportional

##### fl2\_type

*type of flow2*

*Continues on next page*



Data type: num

Flow control type for flow2. Permitted values:

- 0 - Flow2 signal not used
- 1 - Constant, speed independent
- 2 - Speed proportional

equip\_no

*equipment number*

Used equipment for this bead. Permitted values: 1-4

gun\_no

*gun number*

Gun on/off signal selection. Allowed values:

Value	Description
0	No on/off signal is activated
1	Gun 1 is activated (single gun)
>1	Selected guns when multiple guns are used: <ul style="list-style-type: none"> <li>• for example 135 means that the guns 1, 3, and 5 are activated.</li> <li>• Max 5 guns can be activated (gun_no = 12345).</li> </ul>

### Limitations

Max number of guns for each pieces of equipment when multiple guns are used: 5

Max number of pieces of equipment: 4

It is not possible to change equipment number within the same sequence of dispense instructions (from Disp\On to Disp\Off).

It is not possible to change from flow control type 2 to flow control type 1 within the same sequence of dispense instructions.

### Predefined data

```
PERS beaddata bead1 := [ " ",100,100,2,2,1,1];
```

Defined in module DPUSER.

The predefined data `bead1` defines flow rate 100 for flow1 and flow2 both of type 2, that is speed proportional flows are used. Only one gun is activated. Equipment number 1 is used. This equipment is defined in equipment data number 1 in the array `equipd` in DPUSER. `bead1` is used as default in the first programmed DispL or DispC instruction.

### Customizing

The RobotWare-Dispense functionality can be customized to adapt to different types of dispensing equipment. For this data type it is possible to delete components if they are not used. It is also possible to add some components and give components own user defined names. See [Customizing RobotWare-Dispense on page 89](#).

Continues on next page

## 3 RAPID reference

---

### 3.2.1 beaddata

*Continued*

---

#### Default structure

```
< dataobject of beaddata >
  <info of string>
  <flow1 of num>
  <flow2 of num>
  <fl1_type of num>
  <fl2_type of num>
  <equip_no of num>
  <gun_no of num>
```

---

#### Related information

	Described in:
Dispensing instructions	<a href="#">DispL/DispC on page 29</a>
Definition of <code>equipdata</code>	<a href="#">equipdata on page 43</a>
Customizing the functionality	<a href="#">Customizing RobotWare-Dispense on page 89</a>
System module DPUSER	<a href="#">System modules on page 49</a>

## 3.2.2 equipdata

### Usage

`equipdata` is used in dispensing applications to hold equipment specific data. This is dispensing data which does not normally vary between different beads. `equipdata`, for each piece of equipment used, is stored in the array `equipd` in the system module `DPUSER`.

### Description

`equipdata` has the following default contents:

- Information string (80 char).
- Reference speed for the scale calculations.
- Acceleration and deceleration values.
- Anticipated times for the digital gun on/off signals.
- Anticipated times for the analog flow signals.
- Equipment delay compensations.
- Flow rate correction factors.

### Arguments

#### Info

*data information*

Data type: `string`

String for external use, maximum 80 characters.

#### on\_time

*gun on anticipate*

Data type: `num`

Predicted time in seconds for activation of the digital gun on/off signals.

#### off\_time

*gun off anticipate*

Data type: `num`

Predicted time in seconds for deactivation of the digital gun on/off signals.

#### ref\_speed

*reference speed*

Data type: `num`

Reference speed in mm/s. Normally 10 - 20% more than the max. dispensing speed used for the equipment in question. Is used in the scale calculation for speed proportional signals. This value must be > 0. See [Calculation of flow values on page 35](#).

#### acc\_max

*maximum acceleration*

Data type: `num`

*Continues on next page*

### 3 RAPID reference

---

#### 3.2.2 equipdata

*Continued*

The absolute value of the limitation in  $\text{m/s}^2$ . The TCP acceleration along the path is limited from the beginning to the end of the bead. If the value is set to -1 this function is deactivated.

decel\_max

*maximum deceleration*

Data type: num

The absolute value of the limitation in  $\text{m/s}^2$ . The TCP deceleration along the path is limited from the beginning to the end of the bead. If the value is set to -1 this function is deactivated.

fl1\_on\_time

*flow1 on anticipate*

Data type: num

Predicted time in seconds for the activation of the flow1 signal when the dispensing instruction is programmed with the \On argument.

fl1\_off\_time

*flow1 off anticipate*

Data type: num

Predicted time in seconds for deactivation of the flow1 signal when the dispensing instruction is programmed with the \Off argument.

If the argument is set to -1 then the automatic reset of the flow signal is deactivated. The signal holds the latest value given from the beaddata in the \Off instruction until start of next bead.

fl1\_inc\_time

*flow1 increase anticipate*

Data type: num

Predicted time in seconds for increasing the flow1 signal at positions where the bead is changed to another.

fl1\_dec\_time

*flow1 decrease anticipate*

Data type: num

Predicted time in seconds for decreasing the flow1 signal at positions where the bead is changed to another.

fl1\_delay

*equipment delay*

Data type: num

Time in seconds to compensate the flow1 signal for the lag in the dispensing equipment when the TCP speed dips, for example at corners.

fl1\_corr

*flow1 correction*

Data type: num

*Continues on next page*

Flow rate correction factor used in the scale calculations for flow1. Normal value: 100 (%)

fl2\_on\_time

*flow2 on anticipate*

Data type: num

Anticipated time in seconds for the activation of the flow2 signal when the dispensing instruction is programmed with the \On argument.

fl2\_off\_time

*flow2 off anticipate*

Data type: num

Anticipated time in seconds for the deactivation of the flow2 signal when the dispensing instruction is programmed with the \Off argument.

If the argument is set to -1 then the automatic reset of the flow signal is deactivated. The signal holds the latest value given from the beaddata in the \Off instruction until start of next bead.

fl2\_inc\_time

*flow2 increase anticipate*

Data type: num

Anticipated time in seconds for increasing the flow2 signal at positions where the bead is changed to another.

fl2\_dec\_time

*flow2 decrease anticipate*

Data type: num

Anticipated time in seconds for decreasing the flow2 signal at positions where the bead is changed to another.

fl2\_delay

*equipment delay*

Data type: num

Time in seconds to compensate the flow2 signal for the lag in the dispensing equipment when the TCP speed dips, for example at corners.

fl2\_corr

*flow2 correction*

Data type: num

Flow rate correction factor used in the scale calculations for flow2. Normal value: 100 (%)

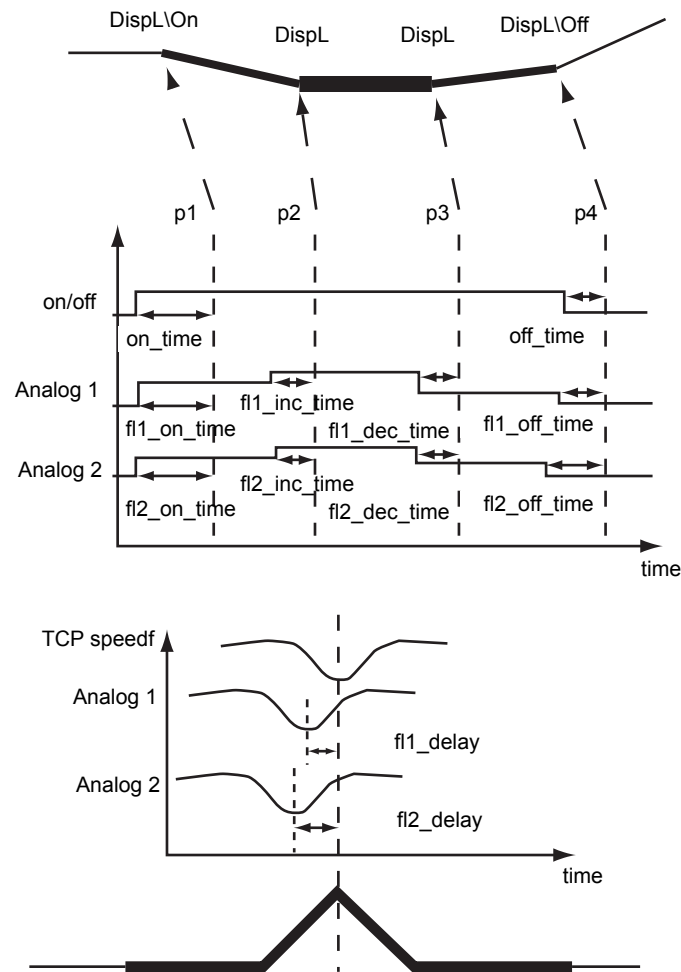
Continues on next page

### 3 RAPID reference

#### 3.2.2 equipdata

Continued

#### Time component description



xx120000584

#### Limitations

Typical repeat accuracy values for the digital (gun on/off) dispense signal settings (CAN/DeviceNet): +/- 1 ms.

Max equipment delay compensation (*fl1\_delay* and *fl2\_delay*) is 0.5 seconds but to get a larger compensation than the servo lag, the system parameter *EventPresetTime* must be increased to a corresponding level. This system parameter can be found in the topic *Motion*, type *Motion Planner*. A run time error is generated if the *EventPresetTime* is too small. Max *EventPresetTime* is 0.5 s, but to not lose cycle time, do not use larger *EventPresetTime* than necessary.

Normally all time components have positive values but it is possible to use also negative values for the time components except for the *fl1\_delay* and *fl2\_delay*. Negative values are limited to 100 ms.

Continues on next page

---

**Recommendations for best accuracy**

It can be necessary to change the *EventPresetTime* also for the other time components, to get best accuracy when large predicted times are used. If some predicted times are larger than 50 ms, we recommend setting the *EventPresetTime* to the same value as the maximum used predicted time.

In some special situations it is not possible to activate the gun signals in right position. This can occur for example if the distance between programmed dispense positions are short in combination with high speeds. However, it is possible to supervise these situations during the program test by activating the configuration parameter *TimeEventSupervision*. This system parameter also belongs to the type *Motion Planner*, in the topic *Motion*.

---

**Predefined data**

```
PERS equipdata equipd{4} :=
  [{" ",0,0,1000,-1,-1,0,0,0,0,0,100,0,0,0,0,0,100]
  [{" ",0,0,1000,-1,-1,0,0,0,0,0,100,0,0,0,0,0,100]
  [{" ",0,0,1000,-1,-1,0,0,0,0,0,100,0,0,0,0,0,100]
  [{" ",0,0,1000,-1,-1,0,0,0,0,0,100,0,0,0,0,0,100]};
```

This array of `equipdata` is predefined in the module `DPUSER` but the data components are intended to be changed by the user to define the actual equipment.

---

**Customizing**

The RobotWare-Dispense package provides opportunities for the user to customize the functionality to adapt to different types of dispensing equipment. For this data type it is possible to delete components if they are not used. It is also possible to add some components and give components own user defined names.

See [Customizing RobotWare-Dispense on page 89](#).

---

**Default structure**

```
< dataobject of equipdata >
  <info of string>
  <on_time of num>
  <off_time of num>
  <ref_speed of num>
  <acc_max of num>
  <decel_max of num>
  <f11_on_time of num>
  <f11_off_time of num>
  <f11_inc_time of num>
  <f11_dec_time of num>
  <f11_delay of num>
  <f11_corr of num>
  <f12_on_time of num>
  <f12_off_time of num>
  <f12_inc_time of num>
  <f12_dec_time of num>
  <f12_delay of num>
  <f12_corr of num>
```

*Continues on next page*

### 3 RAPID reference

---

#### 3.2.2 equipdata

*Continued*

---

#### Related information

	Described in:
Dispensing instructions	<a href="#">DispL/DispC on page 29</a>
Definition of beaddata	<a href="#">beaddata on page 40</a>
Customizing the functionality	<a href="#">Customizing RobotWare-Dispense on page 89</a>
System module DPUSER	<a href="#">System modules on page 49</a>



## 4 System modules

### 4.1 DPUSER

#### Introduction

The system module DPUSER contains predefined data and routines for the dispense application. The module is intended for the purpose of changing and customizing the behavior of the dispense functionality.

#### Data types

The following data types are predefined in DPUSER:

beaddata	<a href="#">beaddata on page 40</a>
equipdata	<a href="#">equipdata on page 43</a>

For `beaddata` and `equipdata` it is possible to reduce the data types by deleting data components that are not used. It is also possible to change the names of the components. See [Customizing RobotWare-Dispense on page 89](#).



#### Note

If the definition of a data type is changed, the routine `dp_set_int_data` must also be changed. See [Routines on page 50](#).

#### Global data

The following global data is predefined in DPUSER. The names of the described data are used internally when a `DispL` or `DispC` instruction is used. Therefore the names must not be changed.

Name	Declaration	Description
<code>dp_fl1_corr</code>	<code>PERS num dp_fl1_corr:= 100</code>	Global override for flow1 signal. All equipment are influenced. Range: 0-200%
<code>dp_fl2_corr</code>	<code>PERS num dp_fl2_corr:= 100</code>	Global override for flow2 signal. All equipment are influenced. Range: 0-200%
<code>dp_dry</code>	<code>PERS bool dp_dry :=FALSE</code>	Used for test in dry mode. If TRUE: No dispense signals are activated. In a MultiMove system this will affect all dispense robots.
<code>bead1</code>	<code>PERS beaddata</code> <a href="#">beaddata on page 40</a>	Predefined <code>beaddata</code> with default values. Used as default in the first programmed <code>DispL/DispC</code> instruction.
<code>equipd</code>	<code>PERS equipdata</code> <a href="#">equipdata on page 43</a>	Predefined array of <code>equipdata</code> with <code>equipdata</code> for equipment 1 - 4. Used when <code>DispL/DispC</code> instructions are executed.
<code>dp_fl_corr_en</code>	<code>CONSTdp_fl_corr_en:= TRUE</code>	If this data is set to FALSE then the possibility to change the global flow correction factors from the Dispense MMI is disabled.

## 4 System modules

### 4.2 Routines

### 4.2 Routines

#### Description

There are some predefined routines installed with the application. They are all called from the dispense kernel and therefore the names must not be changed. Some of the routines have no default functionality.



#### Note

Many of the routines described below are executed during the motion phase. If there is too much time consuming code in these routines, the maximum possible dispensing speed will be reduced.

#### Data setup routine

PROC dp_set_int_data	This routine is used to connect user defined data component names to names used internally. The routine is called when a DispL or DispC instruction is executed. The routine must be changed if the definition of the dispensing data types is changed.
----------------------	---

#### Flow calculation routines

FUNC dp_calcf1_type1	This routine is used to calculate the flow1 logical value when flow type = 1.
FUNC dp_calcf1_type2	This routine is used to calculate the flow1 logical value when flow type = 2.
FUNC dp_calcf2_type1	This routine is used to calculate the flow2 logical value when flow type = 1.
FUNC dp_calcf2_type2	This routine is used to calculate the flow2 logical value when flow type = 2.

For more information about the flow calculations, see [Calculation of flow values on page 35](#).

#### Event routines

PROC dp_err_actions	This routine is executed when an error detected by the RobotWare-Dispense occurs.
PROC dp_power_on	This routine is executed each time the system is switched on.
PROC dp_start	The routine is executed each time the execution of a program is started from the beginning. (From Main)
PROC dp_restart	The routine is executed each time the execution of a stopped program is continued.
PROC dp_restart_proc	The routine is executed during restart of an interrupted dispense sequence, just before the process signals are activated.
PROC dp_stop	This routine is executed when a normal stop is performed during program execution.
PROC dp_qstop	This routine is executed when an emergency stop is performed during program execution.

## 5 System parameters

### 5.1 Topic I/O

#### Introduction

The RobotWare-Dispense package can be configured to suit different types of process equipment. This chapter describes the default defined signals used by RobotWare-Dispense and their dependency on the equipment. This signal configuration is enough for running the default version of the RobotWare-Dispense package.

Dispense signals for four equipment are predefined. During installation the predefined signals that are used must be connected to the corresponding physical equipment. It is not required to use all signals.



#### Note

The digital gun on/off signals for each equipment are internally activated as a signal group. This group must be connected to the physical signal (or signals) for the gun and the signals must follow the rules for group signals, see *Operating manual - IRC5 with FlexPendant*.

#### Output signals controlling dispensing equipment 1

Signal	Type	Description
doEqu1Gun1	digital output	Signal to open/close gun 1.
doEqu1Gun2	digital output	Signal to open/close gun 2 (if used).
doEqu1Gun3	digital output	Signal to open/close gun 3 (if used).
doEqu1Gun4	digital output	Signal to open/close gun 4 (if used).
doEqu1Gun5	digital output	Signal to open/close gun 5 (if used).
goEqu1Guns	output group	Group used to activate the digital gun signals for equipment 1. Internally activated. This output group must be connected to the physical signals for the used guns.
doEqu1Active	digital output	A high signal indicates that the dispensing process is active.
doEqu1Err	digital output	This signal is activated when an internal dispense error is generated.
doEqu1OvrSpd	digital output	A high signal indicates that the calculated value for one analog output signal exceeds the logical maximum value.
aoEqu1F1	analog output	Analog signal for flow1.
aoEqu1F2	analog output	Analog signal for flow2.
goEqu1SwitchNo	output group	This group can be used to set a program number or switch point number to the dispense equipment, see <a href="#">Customizing RobotWare-Dispense on page 89</a> .

*Continues on next page*

## 5 System parameters

---

### 5.1 Topic I/O

*Continued*

---

#### Output signals controlling dispensing equipment 2-4

Same as above but the equipment numbers are changed in the signal names.

---

#### Other signals

Signal	Type	Description
doBwdOnPath	digital output	Used for the bwd on path function if restart type 1 is activated and dp_bwd_key > 0. This signal is cross connected to the digital input <i>diBwdOnPath</i> .
diBwdOnPath	digital input	Used for the bwd on path function if restart type 1 is activated and dp_bwd_key > 0. Internally used.

---

#### Analog output scaling

The analog output signals above are defined with the following default values:

- Logical Min: 0
- Logical Max: 100
- Physical Min: 0 V
- Physical Max: 10 V

Normally, you only need to adapt the Physical Min and Max to the equipment in use. See [Calculation of flow values on page 35](#).

## 5.2 Topic Process

### 5.2.1 Type Dispense GUI

#### 5.2.1.1 The Dispense GUI type

---

##### Overview

This section describes the type *Dispense GUI* which belongs to the topic *Process*. Each parameter of this type is described in a separate information topic in this section.

---

##### Cfg name

DISPENSE\_GUI

---

##### Type description

The *Dispense GUI* type contains a number of parameters that defines the characteristics for the graphical user interface. There is one set of parameters of the type *Dispense GUI* for each equipment.

## 5 System parameters

---

### 5.2.1.2 Equipment

#### 5.2.1.2 Equipment

---

**Parent**

*Equipment* belongs to the type *Dispense GUI*, in the topic *Process*.

---

**Cfg name**

name

---

**Description**

*Equipment* is used to define the equipment name in the system parameters. To change the equipment name that is visible to the user, see [Equipment name on page 56](#).

---

**Limitations**

The parameter is visible but not editable in the software configuration tools. It is not allowed to change the name from the configuration file.

---

**Allowed values**

*Equipment\_1*  
*Equipment\_2*  
*Equipment\_3*  
*Equipment\_4*

#### 5.2.1.3 Equipment visible

---

**Parent**

*Equipment visible* belongs to the type *Dispense GUI*, in the topic *Process*.

---

**Cfg name**

equipment\_visible

---

**Description**

*Equipment visible* defines if the equipment should be visible from the graphical user interface.

---

**Usage**

Set the value to No to hide the equipment in the graphical user interface.

---

**Allowed values**

Yes or No.

Default value is Yes.

## 5 System parameters

---

### 5.2.1.4 Equipment name

#### 5.2.1.4 Equipment name

---

**Parent**

*Equipment name* belongs to the type *Dispense GUI*, in the topic *Process*.

---

**Cfg name**

equipment\_name

---

**Description**

*Equipment name* defines the equipment name in the graphical user interface.

---

**Allowed values**

A string with maximum 80 characters.



#### 5.2.1.5 Flow1 visible

---

**Parent**

*Flow1 visible* belongs to the type *Dispense GUI*, in the topic *Process*.

---

**Cfg name**

flow1\_visible

---

**Description**

*Flow1 visible* defines if the flow meter 1 should be visible from the graphical user interface.

---

**Allowed values**

Yes or No.

Default value is Yes.

## 5 System parameters

---

### 5.2.1.6 Flow1 name

#### 5.2.1.6 Flow1 name

---

**Parent**

*Flow1 name* belongs to the type *Dispense GUI*, in the topic *Process*.

---

**Cfg name**

flow1\_name

---

**Description**

*Flow1 name* defines the name of flow meter 1 in the graphical user interface.

---

**Allowed values**

A string with maximum 80 characters.

#### 5.2.1.7 Flow2 visible

---

**Parent**

*Flow2 visible* belongs to the type *Dispense GUI*, in the topic *Process*.

---

**Cfg name**

flow2\_visible

---

**Description**

*Flow2 visible* defines if the flow meter 2 should be visible from the graphical user interface.

---

**Allowed values**

Yes or No.

Default value is Yes.

## 5 System parameters

---

### 5.2.1.8 Flow2 name

#### 5.2.1.8 Flow2 name

---

**Parent**

*Flow2 name* belongs to the type *Dispense GUI*, in the topic *Process*.

---

**Cfg name**

flow2\_name

---

**Description**

*Flow2 name* defines the name of flow meter 2 in the graphical user interface.

---

**Allowed values**

A string with maximum 80 characters.

## 5.2.2 Type Dispense Restart

### 5.2.2.1 The Dispense Restart type

---

#### Overview

This section describes the type *Dispense Restart* which belongs to the topic *Process*. Each parameter of this type is described in a separate information topic in this section.

---

#### Cfg name

DISPENSE\_RESTART

---

#### Type description

The *Dispense Restart* type contains a number of parameters that defines the characteristics for restarting an interrupted dispense process. There is one set of parameters of the type *Dispense Restart* for each equipment.

---

#### Related information

For more information, see [Using the Dispense Restart function on page 25](#).

## 5 System parameters

---

### 5.2.2.2 Equipment

#### 5.2.2.2 Equipment

---

**Parent**

*Equipment* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

name

---

**Description**

*Equipment* is used to define the equipment name in the system parameters. To change the equipment name that is visible to the user, see [Equipment name on page 56](#).

---

**Limitations**

The parameter is visible but not editable in the software configuration tools. It is not allowed to change the name from the configuration file.

---

**Allowed values**

*Equipment\_1*  
*Equipment\_2*  
*Equipment\_3*  
*Equipment\_4*

---

### 5.2.2.3 Restart type

---

**Parent**

*Restart type* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

restart\_type

---

**Description**

*Restart type* is used to set the restart behavior for the equipment. It is possible to set three different restart levels for each equipment.

0	Restart is disconnected. (Default). The process is started in the next bead, i.e. in the next <code>Disp\On</code> instruction.
1	Restart of current bead with possibilities to select wet or dry.
2	Restart of current bead in a no query mode (without questions)

---

**Allowed values**

A value between 0 and 2.

The default value is 0.

## 5 System parameters

---

### 5.2.2.4 Gun on anticipate

#### 5.2.2.4 Gun on anticipate

---

**Parent**

*Gun on anticipate* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

gun\_on\_anticipate

---

**Description**

*Gun on anticipate* is used to set the preaction time for the digital gun signal related to the start of motion.

---

**Allowed values**

A value between -1 and 1, specifying the preaction time in seconds.  
The default value is 0.



#### 5.2.2.5 Flow anticipate

---

**Parent**

*Flow anticipate* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

flow\_anticipate

---

**Description**

*Flow anticipate* is used to set the preflow time for the analog signals related to the start of motion.

---

**Allowed values**

A value between 0 and 1, specifying the preflow time in seconds.  
The default value is 0.5.

## 5 System parameters

---

### 5.2.2.6 Preflow value flow1

#### 5.2.2.6 Preflow value flow1

---

**Parent**

*Preflow value flow1* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

preflow\_value\_flow1

---

**Description**

*Preflow value flow1* is used to set the preflow value for the analog signal 1.

This value is used when the signal is TCP speed proportional and activated before start of motion.

The value is a percentage value related to the logical max value for the used signal.

---

**Allowed values**

A value between 0 and 100%.

The default value is 50%.

#### 5.2.2.7 Preflow value flow2

---

**Parent**

*Preflow value flow2* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

preflow\_value\_flow2

---

**Description**

*Preflow value flow2* is used to set the preflow value for the analog signal 2.

This value is used when the signal is TCP speed proportional and activated before start of motion.

The value is a percentage value related to the logical max value for the used signal.

---

**Allowed values**

A value between 0 and 100%.

The default value is 50%.

## 5 System parameters

---

### 5.2.2.8 Backward key

#### 5.2.2.8 Backward key

---

**Parent**

*Backward key* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

bwd\_key

---

**Description**

*Backward key* is used to activate a programmable key for the backward on path function.

---

**Allowed values**

A value between 0 and 4.  
The default value is 0 (not used).

#### 5.2.2.9 Backward distance after stop

---

**Parent**

*Backward distance after stop* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

bwd\_distance\_after\_stop

---

**Description**

*Backward distance after stop* is used to set the backward distance before restart after a program stop.

---

**Allowed values**

A value between 0 and 100, specifying the backward distance in mm.  
The default value is 0.

---

## 5 System parameters

---

### 5.2.2.10 Backward distance after quick stop

#### 5.2.2.10 Backward distance after quick stop

---

**Parent**

*Backward distance after quick stop* belongs to the type *Dispense Restart*, in the topic *Process*.

---

**Cfg name**

bwd\_distance\_after\_qstop

---

**Description**

*Backward distance after quick stop* is used to set the backward distance before restart after a quick stop.

---

**Allowed values**

A value between 0 and 100, specifying the backward distance in mm.  
The default value is 0.

## 5.2.3 Type Dispense Signals

### 5.2.3.1 The Dispense Signals type

---

#### Overview

This section describes the type *Dispense Signals* which belongs to the topic *Process*. Each parameter of this type is described in a separate information topic in this section.

---

#### Cfg name

DISPENSE\_SIGNALS

---

#### Type description

The *Dispense Signals* type contains a number of parameters that defines which signals, configured in topic *I/O*, that are used as dispense signals. There is one set of parameters of the type *Dispense Signals* for each equipment.

## 5 System parameters

---

### 5.2.3.2 Equipment

#### 5.2.3.2 Equipment

---

**Parent**

*Equipment* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

name

---

**Description**

*Equipment* is used to define the equipment name in the system parameters. To change the equipment name that is visible to the user, see [Equipment name on page 56](#).

---

**Limitations**

The parameter is visible but not editable in the software configuration tools. It is not allowed to change the name from the configuration file.

---

**Allowed values**

*Equipment\_1*  
*Equipment\_2*  
*Equipment\_3*  
*Equipment\_4*



#### 5.2.3.3 Flow1

---

**Parent**

*Flow1* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

ao\_flow1

---

**Description**

Analog signal for flow 1.

---

**Allowed values**

All configured analog output signals.

## 5 System parameters

---

### 5.2.3.4 Flow2

### 5.2.3.4 Flow2

---

**Parent**

*Flow2* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

ao\_flow2

---

**Description**

Analog signal for flow 2.

---

**Allowed values**

All configured analog output signals.

### 5.2.3.5 OnOff

---

**Parent**

*OnOff* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

go\_onoff

---

**Description**

Signal group used to activate the digital gun signal or signals.

---

**Allowed values**

All configured digital output signal groups.

## 5 System parameters

---

### 5.2.3.6 Dispense active

#### 5.2.3.6 Dispense active

---

**Parent**

*Dispense active* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

do\_active

---

**Description**

A digital signal that indicates that the dispensing process is active.

---

**Allowed values**

All configured digital output signals.

#### 5.2.3.7 Dispense error

---

**Parent**

*Dispense error* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

do\_error

---

**Description**

A digital signal that indicates that an internal dispense error is generated.

---

**Allowed values**

All configured digital output signals.

## 5 System parameters

---

### 5.2.3.8 Overspeed

#### 5.2.3.8 Overspeed

---

**Parent**

*Overspeed* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

do\_overspeed

---

**Description**

A digital signal that indicates that the calculated value for one of the analog flow signals reached it's maximum value.

---

**Allowed values**

All configured digital output signals.

#### 5.2.3.9 Switch value

---

**Parent**

*Switch value* belongs to the type *Dispense Signals*, in the topic *Process*.

---

**Cfg name**

go\_switch

---

**Description**

Signal group that can be used to set a program number or switch point number to the dispense equipment.

For more information, see [How to use an extra signal group with bead information to the process equipment on page 93](#).

---

**Allowed values**

All configured digital output signal groups.

**This page is intentionally left blank**



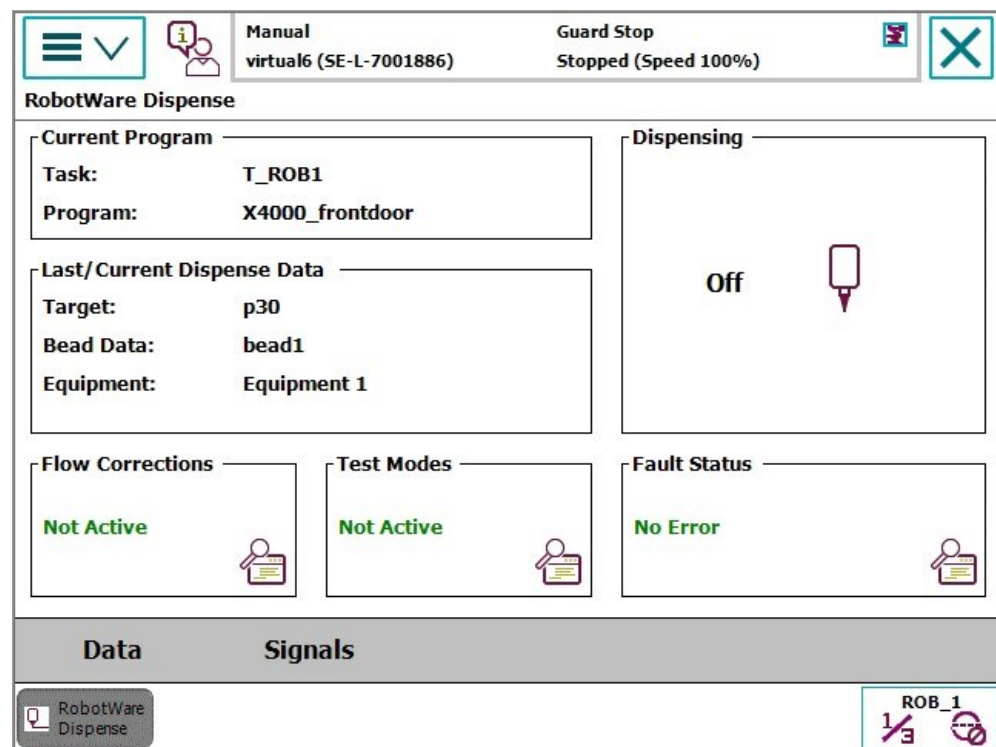
## 6 FlexPendant Interface

### 6.1 Application overview

#### Introduction

The RobotWare Dispense operator interface, in this chapter referred to as Dispense GUI, is available from the FlexPendant ABB menu. It is a graphical interface addressing dispense users, designed to fulfill their specific needs. Dispense related information is presented in an instructing way, enabling operators to easily and quickly get their every day tasks done.

FlexPendant offers considerable flexibility regarding how to do things. Actually, all functionality provided by Dispense GUI can also be found elsewhere. It needs to be pointed out that advanced dispense users will have to use other FlexPendant applications or RobotStudio to perform certain tasks. In this manual, when appropriate, alternative ways of doing things will be indicated. For further information on using the FlexPendant, see *Operating manual - IRC5 with FlexPendant*.



xx1200000136

#### Topics

The Dispense GUI covers following topics:

- **Main View**
- **Process Data**
- **Process Signals**

*Continues on next page*

## 6 FlexPendant Interface

---

### 6.1 Application overview

*Continued*

#### Main View

**Main View** provides basic information about the currently executing dispense program with possibilities to reach other views and sub views.

#### Process Data

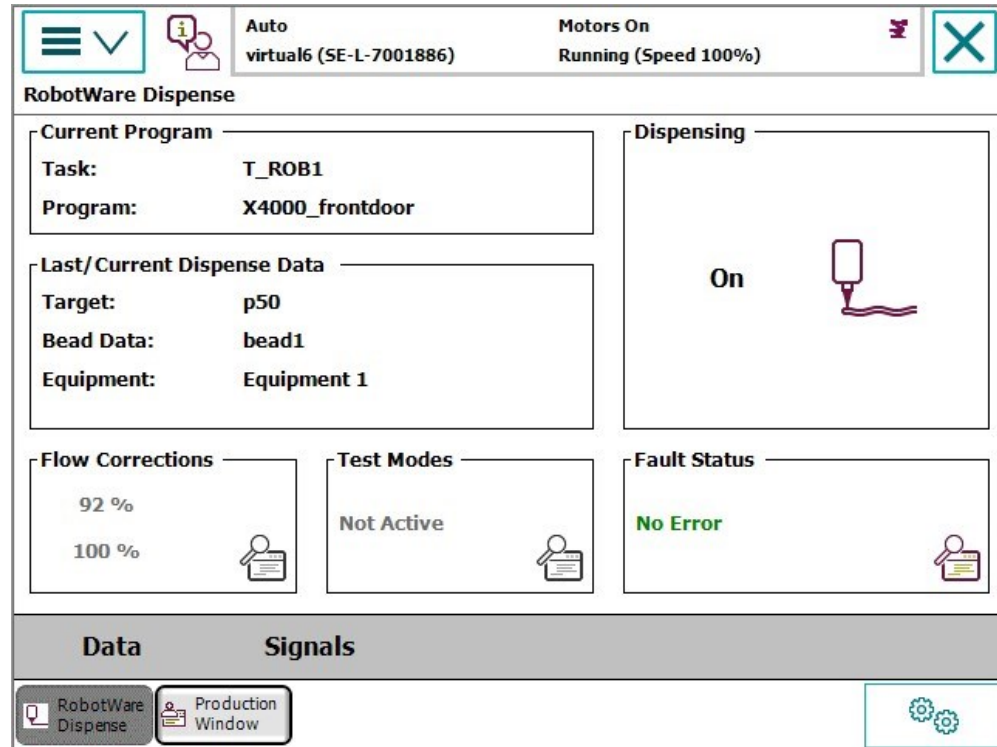
**Process Data** presents dispense specific data types and lists all instances of a selected data type, thus offering a quick and easy way to edit any `beaddata` or `equipdata` in order to tune the system for improved performance.

#### Process Signals

**Process Signals** displays the most important output signals controlling the dispense process equipment. Digital and analog signals are shown in an illustrative way, reflecting the real equipment of the specific dispense system in use. When testing the system in manual mode, it is possible not only to view but also to activate the process signals. See [Process Signals on page 87](#).

## 6.2 Main View

### Description



xx120000135

### Current program

In the preceding figure, the currently executing program is *X4000\_frontdoor*. In a single system the executing task will always be T\_ROB1. In a MultiMove system, however, with several robots executing different dispense programs, the different robot tasks are available from a **Task** menu in the command bar.

### Last/Current dispense data

This area displays the name of current or latest dispense target, the `beaddata` used in the current or latest executed dispense instruction and current or latest used equipment number.

*Continues on next page*

## 6 FlexPendant Interface




### 6.2 Main View

Continued

#### Dispensing

The preceding figure, shows the Main View when the process is active. The selected robot is currently executing a dispense instruction as illustrated by the dispense gun icon. When the execution reaches a `DispL\OFF` instruction the icon will shift to **Off** appearance.

If the system is run in simulated mode the **Dry mode** icon will remain during program execution, as dispensing is never activated.

 xx1200000141	On	The robot is currently dispensing.
 xx1200000142	Off	The robot is stopped or moving to a position where dispensing will start.
 xx1200000140	Dry mode	Simulated mode. The RAPID variable <code>dp_dry</code> is set to true, which means the dispensing functionality is blocked out.

#### Flow corrections

Information in this area tells if any flow correction is active or not. Tutching this area in manual stopped mode will open a subview with possibilities to see the actual flow corrections and also change the global flow corrections for all pieces of equipment. The global variable `dp_fl1_corr` (defined in `DPUSER.sys`) has in this example been changed to 92%, thus affecting flow 1 for all pieces of equipment in the system.

The possibility to change the flow corrections from this view can be deactivated. See the description of the data `dp_fl_corr_en` in [System modules on page 49](#).

#### Test modes

Information in this area tells if the test mode for dry testing (dry mode) is active or not. Tutching this area in manual stopped mode will open a subview with possibilities to activate/deactivate dry mode. The global variable `dp_dry`, defined in `DPUSER.sys`, will be affected.

When dry mode is selected this is also indicated with a small icon in the upper right corner in all dispense views.

#### Fault Status

Information in this area shows if there is a new Dispense error. Tutching this area will open a list with all the Dispense errors from the error log, with possibilities to get more information about the problems. The error information in the Main View will be cleared when a new Disp instruction is executed.

## 6.3 Process Data

### 6.3.1 Introduction

#### Description

The **Process Data** view offers a quick way to modify dispense data.

RobotWare Dispense - Process Data

Data type: **beaddata**  
Task: **T\_ROB1**

Select the data you want to view or edit.

Name ▲	Value	Module
bead1	["",100,100,2,2,1,1]	DPUSER
bead2	["",50,80,2,2,1,6]	TSTXXD
bead20	["20 mm",40,40,2,2,1,1]	TSTXXD
bead40	["40 mm",80,80,2,2,1,23]	TSTXXD
nobead	["",40,40,2,2,1,0]	TSTXXD

Edit Value    Data Type ▲    Refresh    Task ▲    Close

Production Window    RobotWare Dispense    ⚙️

xx1200000132

#### Data types

The data types available from the **Data Type** menu are `beaddata` and `equipdata`. The types are described in [Dispense data types on page 15](#).

#### Tasks and MultiMove

In a MultiMove system you can choose to view or edit data of any dispense task, available from the **Task** menu. In [Process Data on page 85](#) all `beaddata` instances of the selected task `T_ROB1` are listed. For `beaddata` it sometimes makes sense to select a robot task, but for the other data types available, selecting a robot task is not really needed. The reason is that all data having to do with equipment will be shared in all tasks (persistent data), just as the physical pieces of equipment are shared among the robots. In a single system there can be up to 4 pieces of equipment. A MultiMove system has the same restriction; up to 4 pieces of equipment can be put in one cell and shared among the robots. Consequently, updating `equipd{2}` in `T_ROB1` will cause an update in any other dispensing task as well.

### 6.3.2 Editing data

---

#### Editing dispense specific data

The Process Data view provides an easy way to modify dispense specific data.

- 1 Tap **Data Type** and select the type to edit.
- 2 For `beaddata` tap **Task** and select task.
- 3 Select the data instance from the list.
- 4 Tap the selected data instance in the list or tap **Edit Value**.
- 5 If the selected data type is an array, tap the element you want to edit.
- 6 Select the component you want to edit in the **Edit Value** dialog.
- 7 Depending on component data type, enter the new value in the numeric or alphabetic pad which will appear.
- 8 Tap **OK** to close the pad.
- 9 Tap **OK** to write the new value to the controller and to leave the **Edit Value** dialog.

If the controller *User Authorization System* (UAS) is applied the user must have the right to edit RAPID programs to be able to edit data, or a message will appear and the controller will not perform the requested operation. To learn more about UAS, see *Operating manual - RobotStudio*.

---

#### Alternative ways

The **Process Data** view is a subset of the functionality offered by the standard FlexPendant **Program Data** window. For dispense users it is most efficient to do as much as they can from the **Process Data** view, and switch to **Program Data** for more advanced data related tasks, such as adding new data or deleting data, for example.



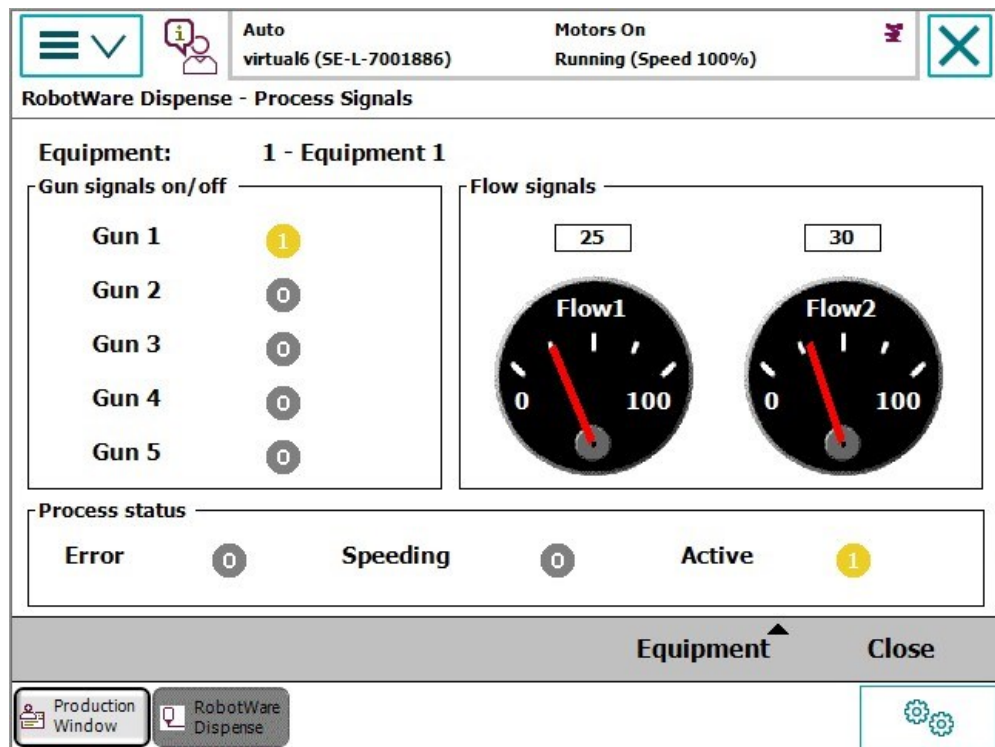
#### Tip

Open the applications you usually work with and use the task bar to switch between them. As many as six different applications can be open at the same time.

## 6.4 Process Signals

### Description

The **Process Signals** view offers a quick way to see the most important dispense signals. In Manual mode it is also possible to change the signal values manually for test purposes.



xx1200000134

### Selected equipment

The selected equipment in the preceding figure is **Equipment 1**. In the **Equipment** menu all pieces of equipment for the specific system are listed and can be viewed, one at a time. Process status signals, as gun and flow signals, belong to selected equipment.

*Continues on next page*

## 6 FlexPendant Interface

### 6.4 Process Signals

*Continued*

#### Displayed signals

Digital signals are shown as 1 (yellow) or 0 (gray). In the figure above one digital gun signal is activated. The number of visible gun signals depends on the size of the output group that is defined for the signals.

Flow signal values are shown in analog meters. Three important status signals are displayed in the **Process status** field. **Error** and **Speeding** indicate that something is wrong and are therefore displayed in red when activated.

Interface name	Default names	Situation when activated
<b>Error</b>	doEqu1Err doEqu2Err doEqu3Err doEqu4Err	An internal dispense error occurs when selected equipment is active. The program may have been stopped in the middle of a bead, for example.
<b>Speeding</b>	doEqu1OvrSpd etcetera	The calculated value for a TCP speed dependent flow signal exceeds the maximum value, which means that the robot's speed is temporarily too high to produce the programmed bead.
<b>Active</b>	doEqu1Active etcetera	Selected equipment is dispensing.

#### Setting new values

In manual stopped mode the appearance of the view is slightly different, displaying buttons for setting I/O signals, when testing the system at installation for example. Tapping such a button for a digital signal instantly sets or resets the signal. To set or reset an analog flow signal, tap **123** and use the numeric pad to enter the new value. The signal value is updated when tapping **OK**.



#### Note

If the controller User Authorization System (UAS) is enabled the user must be authorized to set I/O signals. If an unauthorized user tries to set a signal a message will appear and the controller will not perform the requested operation. To learn more about UAS, see *Operating manual - RobotStudio*.

#### Alternative methods

The general FlexPendant application for viewing and setting I/O signals can be used as an alternative to the **Process Signals** view. Open **Inputs and Outputs** from the ABB menu and tap **View** followed by **Analog** for flow signals, or **View** followed by **Digital Outputs** for gun and status signals.



---

# 7 Customizing RobotWare-Dispense

## 7.1 Introduction

---

### Customizing possibilities

The RobotWare-Dispense functionality can be customized to adapt to different dispensing equipment. One of the purposes of this customizing process is to reduce the amount of data and number of variables presented to the operator.

The following customizing is described in this manual:

- [How to redefine the data types beaddata and equipdata on page 92.](#)
- [How to use your own signal names for internal dispense signals on page 92.](#)
- [How to add functionality in the process sequence on page 92.](#)
- [How to create other equipment specific programming instructions on page 93.](#)
- [How to create service routines for manual functions on page 93.](#)
- [How to use an extra signal group with bead information to the process equipment on page 93.](#)
- [How to make it possible to program the process speed in beaddata on page 94.](#)
- [How to make it possible to program a z offset in beaddata. on page 95.](#)
- [How to change the flow scale calculation algorithm on page 95.](#)
- [How to customize the graphical user interface for Dispense on page 96.](#)
- [How to reduce the number of handled and visible equipment on page 97](#)

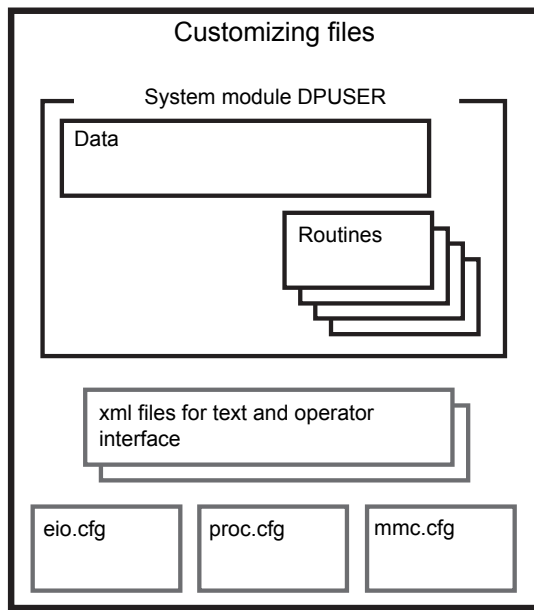
## 7 Customizing RobotWare-Dispense

### 7.2 Files to be changed during the customizing

#### 7.2 Files to be changed during the customizing

##### Description

The customizing is done by changing a number of predefined files, preferably using a PC. The following RAPID modules and configuration files should be changed when customizing RobotWare-Dispense.



xx1400001081

##### DPUSER

DPUSER contains global data and routines affecting the entire program. The module also contains the definition of the dispensing data types, see [System modules on page 49](#).

During installation, and when using the restart mode **Reset system**, the default version of DPUSER is copied to the directory `../HOME/Dispense` in the system. This file is then used during start. If desired, this file can be exchanged to a customized version before start.

##### Text files

Xml files containing text strings for for example the internal error handling: `gldtext.xml` and `gld_elogtext.xml`.

These texts are loaded during installation from the Dispense option directory in the mediapool (`../options/disp/language/en` for the english version of the text files).

##### I/O configuration (eio.cfg)

As default, dispensing output signals are defined for four pieces of equipment but all signals are simulated. The used dispense outputs must be connected to physical signals. Signals not used must as simulated signals (see [Topic I/O on page 51](#)).

*Continues on next page*

---

#### MMC configuration (mmc.cfg)

This configuration file contains for example information about which instructions are included in the different instruction pick lists.

---

#### Process configuration (proc.cfg)

For dispense the process parameters are divided into three types:

Type	Description
Dispense GUI	Parameters for customizing the GUI for each equipment.
Dispense Restart	Parameters to set the start behavior for each equipment.
Dispense Signals	Parameters for connecting the dispense signals for each equipment.

## 7 Customizing RobotWare-Dispense

### 7.3 Customizing guides

### 7.3 Customizing guides

#### How to redefine the data types beaddata and equipdata

It is possible to:

- Add or delete data components.
- Move data components from for example `equipdata` to `beaddata`.
- Change the names of the data components.

Change the definition of the data types in DPUSER to desired.	
Change corresponding instructions in the data setup routine <code>dp_set_int_data</code> in DPUSER.	This routine is used to connect the user defined data components to internal data.
Change the structure and the default values of following data in DPUSER (if corresponding data type is changed):	- PERS beaddata bead1 - PERS equipdata equipd{4}
Change text strings in <code>gld_elogtext.xml</code> if necessary.	

#### How to use your own signal names for internal dispense signals

It is possible to change the names of the predefined dispense outputs and output groups.

Change the signal names in the I/O configuration file.	
Change the corresponding signal in the Process configuration file.	

#### How to add functionality in the process sequence

It is possible to add own code to the different routines in DPUSER. Following routine is executed each time a dispense instruction is executed.

<code>dp_set_intdata</code>	Will be executed in the beginning of each Dispense instruction. The main task for this routine is to move the process data to the kernel, but in some cases it can be appropriate to add some user code to this routine.
-----------------------------	--



#### Note

The routine is executed during the motion phase. If there is too much time consuming code in this routine, the maximum possible dispensing speed will be reduced. See description of the event routines in [System modules on page 49](#).

When code is added to `dp_set_intdata` it is sometimes desirable to use information about current instruction. To get information following data can be used:

<code>current_bead</code>	<code>beaddata</code>	currently used beaddata
<code>int_dp_on</code>	bool	TRUE if <code>/On</code> argument in current instruction.
<code>int_dp_off</code>	bool	TRUE if <code>/Off</code> argument in current instruction.
<code>int_dp_finep</code>	bool	TRUE if fine point in current instruction.

*Continues on next page*

int_dp_posname	string	The name of the currently used <code>robtarget</code> . If the position is stored in the instruction (*) the string is empty.
----------------	--------	---

**Example:** To print log information about current instruction (`robtarget` name and corresponding `beaddata`) in a logfile, the following instruction can be added to `dp_set_intdata`:

```
Write logfile," position: "+int_dp_posname+" beaddata:
"+Argname(current_bead)+" " ;
```



#### Note

The time when the routine `dp_set_int_data` is running is not coordinated with the robot movement. The routine is normally executed before the robot is passing the programmed position before current `Disp` instruction.

If it is important to have a more coordinated routine this can be done by creating a trap routine. Use for example the signal `doEqualActive` for the activation, if the routine shall be executed in the beginning or end of the bead.

#### How to create other equipment specific programming instructions

Create global routines with desired name, syntax and functionality. Use a system module with the attribute `NOVIEW` or `NOSTEPIN`.

The MMC configuration has to be changed to define the instruction syntax and to get the new instructions in an instruction pick list.

#### How to create service routines for manual functions

Create RAPID routines for **Manual Functions** like for example *Move to Purge*, *Move to Home*, *Purge*, *Reload...*

The MMC configuration has to be changed. Add the routine names under `MMC_SERV_ROUT_STRUCT`. These functions are then available from the **Debug** menu in the **Program Editor**.

#### How to use an extra signal group with bead information to the process equipment

The dispense kernel is prepared for activation of an extra signal group a user defined time before the robot is passing a programmed `Disp\On` or `Disp` instruction. This signal group can for example be used to give switch point information or program number information to the process equipment. The group is cleared in `Disp\Off` instructions.

If this functionality is used it is suitable to add a new data component (for example `switch_number`) to `beaddata` and an other data component (for example `switch_time`) to `equipdata`. See [How to redefine the data types beaddata and equipdata on page 92](#).

In the `DPUSER` routine `dp_set_int_data` the internal data `int_dp_data.switch_no` and `int_dp_data.switch_time` has to be updated with information from current `beaddata` and `equipdata`.

Continues on next page

## 7 Customizing RobotWare-Dispense

---

### 7.3 Customizing guides

*Continued*

An output group for each equipment (for example `goEqualSwitchNo`) is already predefined in the default `eio.cfg`, and handled in the *Dispense Signals* type in the `proc.cfg`.

---

#### How to make it possible to program the process speed in beaddata

It is possible to increase the `beaddata` with a speed data component. This speed data will then be used during the process instead of using the speed data programmed in the dispense instruction.

To be able to use this possibility, add a new data component (for example `tcp_speed`) to the `beaddata` RECORD in DPUSER. See [How to redefine the data types beaddata and equipdata on page 92](#).

In the DPUSER routine `dp_set_int_data` the internal data `int_dp_data.tcp_speed` has to be updated with information from current `beaddata`, in a similar way as other data components from `beaddata` are updated.

For example add following instruction:

```
int_dp_data.tcp_speed := current_bead.tcp_speed;
```

Add the new data component to all already programmed `beaddata` to be used.

Remember to add the new component also to the predefined `beaddata bead1` in DPUSER.

More details:

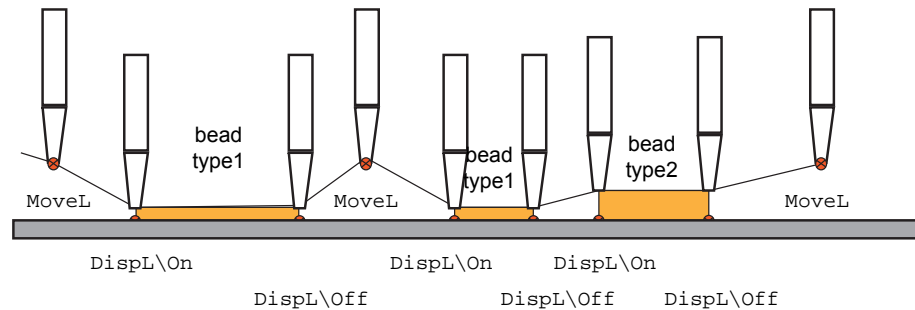
- When the program is executed step wise forward or backward the programmed speed in the instruction is used.
- To reduce unnecessary accelerations or decelerations during the process, the speed data from current `beaddata` will be used also during the `DispL\On` or `DispC\On` segment, before the process is started.
- All speed changes will be affected from the middle of the corner zone, also when the optional argument `\D` is used. It is for that reason recommended to not use the optional instruction argument `\D` when the speed is programmed in `beaddata`.
- The speed from current `beaddata` is also used during test in dry mode.
- If the speed in the `beaddata` is set to -1 this function is disabled. The programmed speed in the instruction will be used in the normal way.

*Continues on next page*

**How to make it possible to program a z offset in beaddata.**

It is possible to increase the `beaddata` with a z offset component. This offset data will then be used during the process as shown in example below.

Example:



xx120000640

During programming all dispense instructions are programmed with the TCP on the sheet or with a constant distance to the sheet. During program execution the tool is automatically lifted (in negative z direction) to correct level for the bead according to the offset value in current `beaddata`.

To be able to use this possibility, add a new data component (for example `z_offset`) to the `beaddata` RECORD in DPUSER. See [How to redefine the data types beaddata and equipdata on page 92](#).

In the DPUSER routine `dp_set_int_data` the internal data `int_dp_data.z_offset` has to be updated with information from current `beaddata`, in a similar way as other data components from `beaddata` are updated.

If the name of the new component is `z_offset`, add following instruction:

```
int_dp_data.z_offset := current_bead.z_offset;
```

Add the new data component to all already programmed `beaddata` to be used.

Remember to add the new component also to the predefined `beaddata bead1` in DPUSER.

More details:

- When dispense positions are changed/modified this shall be done without offset as during programming. Therefore the z offset is omitted when the program is executed step wise forward or backward in Manual mode.
- Minimum and maximum permissible offset is internally limited to 0-20 mm. It is however possible to change these limits using following prepared DPUSER data.

```
CONST num dp_z_offset_max := xx;
```

```
CONST num dp_z_offset_min := xx;
```

**How to change the flow scale calculation algorithm**

Change the predefined calculation algorithm in following event routines in DPUSER:

<code>dp_calcf1_type1:</code>	Used to calculate the flow1 logical value when flow type = 1
<code>dp_calcf1_type2:</code>	Used to calculate the flow1 logical value when flow type = 2
<code>dp_calcf2_type1:</code>	Used to calculate the flow2 logical value when flow type = 1

Continues on next page

## 7 Customizing RobotWare-Dispense

### 7.3 Customizing guides

Continued

dp_calcf2_type2:	Used to calculate the flow2 logical value when flow type = 2
------------------	--

#### How to customize the graphical user interface for Dispense

It is possible to change the names for the equipment and flow signals. It is also possible to change the number of visible digital and analog gun signals.

The **Process Signals** view should mirror the real dispense equipment in use. It is possible to customize the GUI by editing the system parameters in type *Dispense GUI* which belongs to the topic *Process*.

The predefined parameters should be edited to reflect the real equipment. This improves both system performance and ease of use since the FlexPendant only displays information that is relevant for a specific installation.

#### Example

This example shows how to set the parameters to represent the an example installation as shown in the following figure, and in the figure in [Process Signals on page 87](#).

Equipment	1	2	3	4
Equipment visible	TRUE	TRUE	FALSE	FALSE
Equipment name	Equipment 1	Karlssons Flow Control	-	-
Flow1 visible	TRUE	TRUE	-	-
Flow1 name	Flow 1	Pressure	-	-
Flow2 visible	TRUE	FALSE	-	-
Flow2 name	Flow 2	-	-	-

Equipment 1 will be displayed as *Equipment 1* showing five gun signals and two flow signals named *Flow 1* and *Flow 2*, see the figure in [Process Signals on page 87](#).

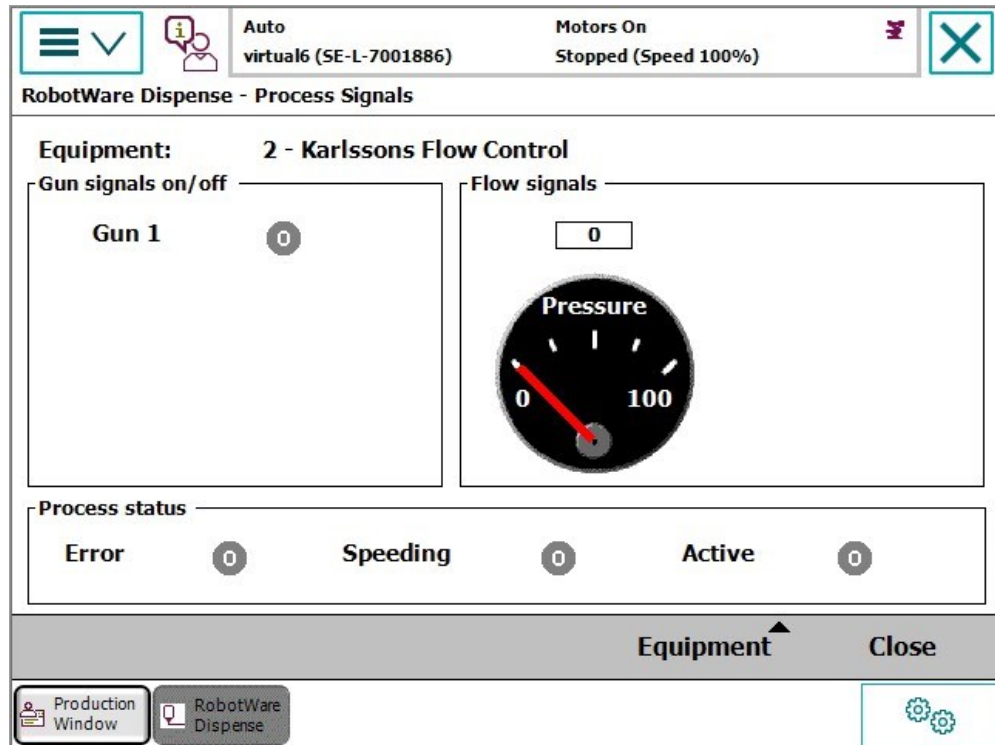
*Equipment 2* will be displayed as *Karlssons Flow Control* showing one flow (named *Pressure*), see the following figure. *Equipment 3* and *Equipment 4*, however, does not exist (the second argument is FALSE), and should therefore not be displayed at all.

As a result the **Equipment** menu will have the following two choices:

- 1 - Equipment 1
- 2 - Karlssons Flow Control

Continues on next page





xx120000133

In this example there is only one signal in the group that is defined for equipment 2.

### How to reduce the number of handled and visible equipment

By default four equipment are visible in data and configuration. This is also the maximum number of equipment to be handled. If less than four equipment is used, it is possible to reduce the number of handled and visible equipment.

	Action
1	Reduce the number of elements in the <code>equipd</code> array in <code>DPUSER</code> . The dispense kernel handles the same number of equipment as the size of this array.
2	Delete all I/O signals for the unused equipment in the system parameters, topic <code>I/O</code> .
3	Delete all system parameters for the unused equipment of the types <code>Dispense GUI</code> , <code>Dispense Restart</code> , and <code>Dispense Signals</code> , which belongs to the topic <code>Process</code> .

**This page is intentionally left blank**

---

## 8 Dispense options

### 8.1 DispensePac Support (901-1)

#### 8.1.1 Introduction

---

##### Description

*DispensePac Support* is an option intended to prepare the system with some basic hardware and software components necessary when different DispensePac function packages are to be used.

ABB DispensePac are function packages for sealing, gluing and similar processes, with some basic hardware and software components common with paint robot systems. These function packages are cost effective solutions for many sealing and dispensing applications. No third party system controller is required.

---

##### Prerequisites

The option *RW Dispense (641-1)* must be installed.

---

##### Contents in DispensePac Support

With *DispensePac Support* the system is prepared with the following basic hardware and software components:

###### Hardware:

- Process Interface Board (PIB) with IPS software, mounted in the cabinet.
- Some harnesses and a switch, also mounted in the cabinet.
- Process I/O (PIO) board(s).

###### Software:

- Software/configuration for communication with PIB and IPS.
- Some RAPID instructions from the paint area, useful also for Dispense applications.

---

##### Default configuration

DispensePac Support uses the default configuration from Dispense. This can then easily be customized to adapt to actual needs in the different DispensePac function packages.

---

##### Available instructions and functions from paint

When *DispensePac Support* is installed following instruction and function from paint are available:

- `SetTmSignal`, see [SetTmSignal on page 100](#).
- `GetSignal`, see [GetSignal on page 103](#).

## 8 Dispense options

---

### 8.1.2.1 SetTmSignal

## 8.1.2 RAPID reference for DispensePac Support

### 8.1.2.1 SetTmSignal

---

#### Usage

`SetTmSignal`, *Set timed signal*, is an instruction that allows digital outputs (DO), group outputs (GO), or analog outputs (AO) to be set with an additional delta time parameter. The delta time says when the signal value should be changed with respect to the previous signal change done by the same instruction.

---

#### Basic example

```
SetTmSignal "AlSolvCC", 0, 0.2;
```

Solvent valve is turned off after 200 ms.

---

#### Arguments

```
SetTmSignal Signal Value dt [\Reset][\Wait]
```

#### Signal

**Data type:** `string`

Identifies the signal to be set.

**Value:** Name of signals of type `signaldo`, `signalgo` or `signalao`.

#### Value

**Data type:** `num`

The desired value of the signal.

#### dt

**Data type:** `num`

Delta time between setting this signal and setting the previous signal.

#### [\Reset]

**Data type:** `bool`

If true, the time reference will be reset.

Recommended to be used on the first instruction in a sequence.

#### [\Wait]

**Data type:** `bool`

If true, the instruction will wait until the signal has been set before continuing.

---

#### Program execution

The instruction will have no timing effect in stepwise forward or backward execution. However, the outputs involved will get the programmed signal levels.



#### Note

There is a separate timing reference for each task, that is timing cannot be related between tasks.

*Continues on next page*

---

**More examples****Example**

A color change sequence might look like follows:

```
SetTmSignal "AlSolvCC", 1, 0.2;
```

First instruction in sequence initiates timing, dt is related to the time when the instruction was executed.

```
SetTmSignal "AlSolvCC", 0, 0.2;
```

Solvent valve is turned off 200ms after it was turned on.

```
SetTmSignal "AlAirCC", 1, 0;
```

Air valve is turned on at the same time the solvent valve is turned off.

```
SetTmSignal "AlAirCC", 0, 0.2;
```

Air valve is turned off after another 200ms.

```
SetTmSignal "AlSolvCC", 1, 0.01;
```

Solvent valve is turned on 10ms after air valve was turned off.

```
SetTmSignal "AlCol", 15, 0.1;
```

Open valve 15 on color change stack.

```
SetTmSignal "AlFluid", 350, 0.05;
```

Start dosing 350ml/min of paint 50ms after color valve was opened.

```
SetTmSignal "AlFluid", 0, 3.1 \Wait;
```

Stop paint dosing after 3.1s and wait until finished.

Note that the different signal types (AO, DO, GO) may be mixed in the same sequence.

This instruction is useful for a graphical color change editor.

---

**Limitations**

The `SetTmSignal` instruction will not be able to do any useful timing when used alone. Only a sequence of instructions will make an accurate timing sequence.

The timing accuracy only works with 'intelligent' nodes, like the IPS node. For other I/O nodes, the signal will be set but with less accurate timing.

The first `SetTmSignal` instruction in a sequence should not be used to operate a time critical signal edge. This is due to the fact that the timing accuracy is relative to the time when the first instruction was executed, and that when starting up such a sequence, there will always be some system delay for the first signal edge to get through. The operational accuracy is depending on that each instruction is allowed to execute some time ahead of the wanted signal transition. When a sequence of such instructions is executed, all instructions will be interpreted immediately (unless the `\Wait` parameter is used), telling the I/O system when to execute each signal transition. Providing that the `dt` parameter is not too small, the I/O system will have time to buffer the signal events, and then it will wait for the time delay(s) to expire.

**Note**

If all timing sequences from the previous instructions have expired by more than 2 seconds, the timing reference for the next `SetTmSignal` instruction will be reset.

*Continues on next page*

## 8 Dispense options

---

### 8.1.2.1 SetTmSignal

*Continued*

---

#### Error handling

When encountering an undefined signal, the execution will be transferred to the nearest error handler with the error `ERR_SIG_NAME`. See example below.

```
SetTmSignal "AlSolvCC", 0, 0.2;
ERROR
IF ERRNO = ERR_SIG_NAME THEN
    error_action;
    RETRY
ENDIF
...
PROC error_action()
    ! error recovery actions
    ...
ENDPROC
```

---

#### Syntax

```
SetTmSignal
[Signal':=']<expression (IN) of string>', '
[Value':=']<expression (IN) of num>', '
[dt':=']<expression (IN) of num>', '
[ '\Reset':=']<expression (IN) of bool >
[ '\Wait':=']<expression (IN) of bool > ';'
```

#### 8.1.2.2 GetSignal

---

##### Usage

`GetSignal` is a function that returns a value of any input or output type.

---

##### Basic examples

```
var string stdoSolvent := "doSolvent";  
num := GetSignal (stdoSolvent);
```

Current value of output *doSolvent* is returned.

---

##### Arguments

```
GetSignal (Signal);
```

##### Signal

**Data type:** string

Identifies the signal to be returned.

**Value:** Name of signals of type `signaldo`, `signalgo` or `signalao`.

##### Return value

**Data type:** num

Returns the value of the signal.

---

##### Program execution

Works in continuous mode and stepwise forwards and backwards execution.

---

##### Limitations

The `GetSignal` instruction will only work for defined I/Os.

Signal name must be given as a variable. Signal name is case sensitive.

---

##### Error handling

When encountering an undefined signal, a warning is issued, and the returned value will be set to -1000 000.

---

##### Syntax

```
<identifier> := GetSignal '('  
[Signal ':='] < expression (IN) of string > ')'';
```

**This page is intentionally left blank**



# Index

## B

beaddata, 40

## C

Change signal names, 92

Create instructions, 93

Create service routines, 93

## D

Data setup routine, 50

default signals, 51

DispC, 29

DispensePac, 99

DispensePac Support, 99

Dispense Restart, 25

DispL, 29

DPUSER, 49

## E

equipdata, 43

Event routines, 50

## F

FlexPendant Interface, 81

Flow calculation routines, 50

Flow corrections, 84

flow scale calculation, 95

## G

GetSignal, 103

Global data, 49

GUI, 96

## L

Limitations, 17

## M

MultiMove, 27

## O

Option content, 12

output signals, 51–52

## P

Process Interface Board (PIB), 99

Process restart, 25

Programming, 17

## R

Redefine data, 92

Restart behavior, 25

Restart function, 25

## S

SetTmSignal, 100

Short beads, 21

signals

    default setup, 51

Simulation, 20

## T

Testing, 20

Test modes, 84

## Z

Z offset, 95







**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 New 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](http://abb.com/robotics)**