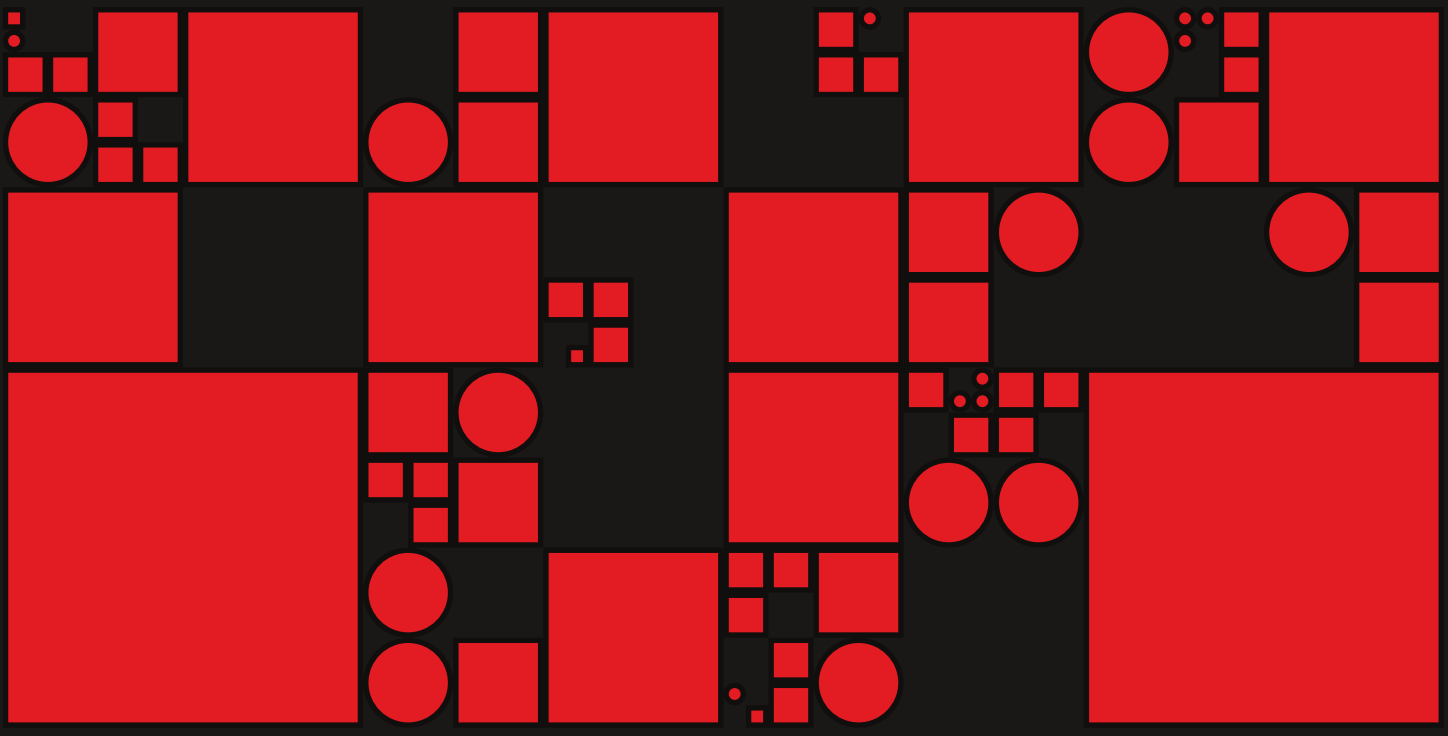
Interface

AutoStore

Operation Manual

Manual revision: 0

Publishing date: 26.09.2022



AS-55126-STD-en
Language: English Original

Imprint

AutoStore AS

Stokkastrandvegen 85

N-5578 Nedre Vats

Norway

Tel: +47 5276 3660

http://www.autostoresystem.com/

Copyright, Intellectual Property, and Disclaimer

All rights reserved. This manual is protected by applicable copyright law and international treaty provisions. AutoStore Technologies AS is the sole proprietary owner of this manual, its contents and related intellectual property, including, but not limited to, creative work, designs, patents and trademarks. No parts of this work, including but not limited to, design, technical solutions, trademarks, graphic, electronic or mechanical, may be reproduced in any form or by any means whether by photocopying, recording, taping, information storage and retrieval systems or otherwise, without the written permission of AutoStore Technologies AS.

This manual and its contents and specifications are subject to change without notice. While every precaution has been taken in the preparation of this document, neither AutoStore Technologies AS nor any other company within the AutoStore Group (affiliates, subsidiaries or parent companies of AutoStore Technology AS) assume any responsibility or liability for errors or omissions, or for any personal injury, loss or damages resulting from the use of or reliance on the information contained in this document or from the use of programs and source code that may accompany it.

AutoStore is a certified product. Modules and parts of the AutoStore system must not be modified, altered, or installed, operated or used in a way not explicitly endorsed by the official documentation. To the fullest extent permissible under applicable law, in no event shall AutoStore Technologies AS or any other company within the AutoStore Group be liable for any claim, loss of profit or any other economic or physical damage caused or alleged to have been caused directly or indirectly by this document.

All specifications provided in this manual follow European standards and may not reflect applicable local standards. If applicable local standards deviate from the ones stated in this manual, then the strictest specifications should be followed.

It is the responsibility of the distributor to ensure that local requirements are followed.

Contents

Imprint . . . 3
Copyright, Intellectual Property, and Disclaimer . . . 4
1. Read this first . . . 9
1.1 About this manual . . . 9
1.2 Conventions and Standards . . . 9
1.3 Contacting us . . . 10
1.4 Signs and Notices . . . 10
1.5 Changes to the Manual . . . 10
1.6 General Safety Information . . . 10
1.7 General Precautions . . . 11
2. Introduction . . . 12
2.1 Task Interface and Bin Interface . . . 12
2.1.1 Task Interface . . . 13
2.1.2 Bin Interface . . . 13
2.1.3 Comparison . . . 14
2.2 Technical Interface . . . 14
2.2.1 XML Message Format . . . 15
2.2.2 Using the AutoStore Interface . . . 16
2.3 Feature Compatibility . . . 16
2.3.1 Features that require AS Router . . . 16
2.3.2 Software Release October 2022 . . . 17
2.3.3 Software Release April 2023 . . . 17
2.3.4 Software Release October 2023 . . . 17
3. Basic Concepts . . . 18
3.1 Bin . . . 18
3.1.1 Bin Properties . . . 18
3.1.2 State . . . 18
3.1.3 Bin Content Code . . . 19
3.1.4 Automatic Preparation of Empty Bins . . . 20
3.2 Port . . . 20
3.2.1 Port Properties . . . 20
3.2.2 Port Mode . . . 21
3.3 Grid . . . 22
4. Task Interface . . . 23
4.1 Basic Concepts . . . 23
4.1.1 Task Group . . . 23
4.1.2 Task . . . 24
4.1.3 Port Properties . . . 24
4.2 Balancing Categories . . . 25
4.3 Performance Guidelines . . . 26
4.4 Port Methods . . . 26
4.4.1 Open Port . . . 26
4.4.2 Open Bin . . . 27
4.4.3 Close Bin . . . 30
4.4.4 Close Port . . . 31
4.4.5 Insert Bin . . . 32
4.4.6 Remove Bin . . . 34
4.4.7 Update Bin . . . 35
4.4.8 Get Port Status . . . 36
4.5 Task Methods . . . 38
4.5.1 Create Taskgroup . . . 38

4.5.2 Update Taskgroup . . . 40
4.5.3 Cancel Taskgroup . . . 42
4.5.4 Add Task . . . 43
4.5.5 Cancel Task . . . 44
4.6 Query Methods . . . 45
4.6.1 Get Bin Information . . . 45
4.6.2 Get Bin Location . . . 47
4.6.3 Get Bin State . . . 49
4.6.4 Get Task Information . . . 51
4.6.5 Get Next Task Groups . . . 53
4.7 Controlling the Sequence of Task Groups . . . 55
4.7.1 Overriding the next Task Group . . . 55
4.7.2 Implementing Cut-Off Time . . . 55
4.7.3 Synchronizing with the Task Group Sequence . . . 56
4.7.4 Specifying a Strict Sequence of Task Groups . . . 56
5. Bin Interface . . . 60
5.1 Basic Concepts . . . 60
5.1.1 Bin Preparation . . . 60
5.1.2 Port Queue . . . 60
5.2 Port Methods . . . 61
5.2.1 Open Port . . . 61
5.2.2 Open Bin . . . 62
5.2.3 Close Bin . . . 64
5.2.4 Close Port . . . 65
5.2.5 Insert Bin . . . 66
5.2.6 Remove Bin . . . 68
5.2.7 Update Bin . . . 69
5.3 Bin Preparation Methods . . . 70
5.3.1 Prepare Bin . . . 70
5.3.2 Insert Prepare Bin . . . 71
5.3.3 Cancel Prepare Bin . . . 72
5.3.4 Set Port Queue . . . 74
5.3.5 Append to Port Queue . . . 75
5.3.6 Begin Group in Port Queue . . . 77
5.3.7 End Group in Port Queue . . . 77
5.4 Query Methods . . . 78
5.4.1 Get Bin Information . . . 78
5.4.2 Get Bin Location . . . 80
5.4.3 Get Bin State . . . 82
5.4.4 Get Preparation Queue . . . 84
5.4.5 Get Port Queue . . . 85
6. Port Processes . . . 87
6.1 Standard Port Flows . . . 87
6.1.1 Putaway Process . . . 87
6.1.2 Picking Process with Task Interface . . . 88
6.1.3 Picking Process with Bin Interface . . . 88
6.2 PickUpPort . . . 89
6.2.1 Get Drawer State . . . 89
6.3 Fusion Port . . . 90
6.3.1 Introduction . . . 91
6.3.2 Pick-To-Light API . . . 91
6.3.3 Show Pick-To-Light . . . 92
6.3.4 Hide Pick-To-Light . . . 94
6.3.5 Compartment Definition . . . 95
6.3.6 Pick-To-Light Example . . . 95
6.4 MultiPort configuration of the Fusion Port . . . 97
6.4.1 Introduction . . . 97
6.4.2 Enable MultiPort . . . 98

6.4.3 Disable MultiPort . . . 98
6.5 Miscellaneous Port Methods . . . 99
6.5.1 Configure Port . . . 99
6.5.2 Get Port Configuration . . . 101
6.5.3 Get Port Weight . . . 102
6.5.4 Flush Port Bin . . . 104
7. Forecast Bin Preparation . . . 106
7.1 Description . . . 106
7.1.1 Bin Mode Forecast . . . 106
7.2 Forecast Methods . . . 106
7.2.1 Forecast Bin . . . 106
7.2.2 Cancel Forecast Bin . . . 108
7.3 Guidelines . . . 109
8. Controlling Bin Locations . . . 111
8.1 Introduction . . . 111
8.1.1 Bin Content Code . . . 111
8.1.2 Cell Content Code . . . 111
8.1.3 Content Code Ranges . . . 112
8.1.4 Interface . . . 112
8.2 Configurations . . . 112
8.2.1 Main Configurations . . . 112
8.2.2 Content Consumers . . . 113
8.3 Interface Methods . . . 114
8.3.1 Update Cell and Bin Content . . . 114
8.4 Managing Frozen Zone . . . 115
8.4.1 Defining the Zones . . . 115
8.4.2 Controlling Bin time outside its Zone . . . 115
9. Port Interface . . . 117
9.1 Port Process . . . 117
9.1.1 Allocate Cell . . . 117
9.1.2 Exchange Bins to and from Grid . . . 117
9.2 Message Flow . . . 118
9.2.1 Port Retrieve Bin from Grid . . . 118
9.2.2 Port Deliver Bin to Grid . . . 118
9.2.3 Port Exchange Bin with Grid . . . 119
9.3 Examples . . . 119
9.3.1 Port with Outgoing Bins to Conveyor Belt . . . 119
9.3.2 Port with Outgoing Bins to Conveyor Belt (mode AUTO) . . . 120
9.3.3 Port with Incoming Bins from Conveyor Belt . . . 121
9.3.4 Port Exchange Bins to and from Grid . . . 121
9.4 Error Handling . . . 125
9.4.1 Failure when Port Retrieves Bin from Grid . . . 125
9.4.2 Failure when Port Deliver Bin To Grid . . . 125
9.5 Port Interface Methods . . . 126
9.5.1 Port Cell Allocate . . . 126
9.5.2 Port Location Ready . . . 127
9.5.3 Port Bin Ready . . . 128
9.5.4 Port Bin mode . . . 130
10. Connecting Grids with Bin Lift . . . 132
10.1 Introduction . . . 132
10.2 Exchanging Bins between the Grids . . . 132
10.3 Transfer Bins to the other Grid . . . 132
11. Miscellaneous Interface Methods . . . 133
11.1 Bin Definition . . . 133
11.1.1 Create Bin . . . 133
11.1.2 Delete Bins . . . 134
11.2 System Configuration . . . 135
11.2.1 Force Charge . . . 135

11.2.2 Set Configuration . . . 136
11.3 Query Methods . . . 138
11.3.1 Get System State . . . 138
12. Log Publisher Interface . . . 140
12.1 Technical Interface . . . 140
12.2 Initial Download . . . 140
12.3 Event Segments . . . 140
12.3.1 Download Start . . . 141
12.3.2 Download Complete . . . 141
12.3.3 Time . . . 141
12.3.4 Time Factor . . . 141
12.3.5 Bin Location . . . 141
12.3.6 Bin Mode . . . 142
12.3.7 Port Location Failed . . . 142
12.3.8 Port Mode . . . 142
12.3.9 Port Definition . . . 142
12.3.10 Cell Lock . . . 142
12.3.11 Lift Cell Lock . . . 143
12.3.12 Port Error . . . 143
12.3.13 Robot Charging . . . 143
12.3.14 Robot Error . . . 143
12.3.15 Robot Unavailable . . . 143
12.3.16 Port Task Groups . . . 143
12.3.17 Pick To Light . . . 144
12.3.18 Drawer State . . . 144
12.3.19 System Mode . . . 144
12.3.20 Alert . . . 144
12.4 Value Codes . . . 144
13. AutoStore Internals and Configurations . . . 147
13.1 Port Carousel Behavior . . . 147
13.2 Port Configurations . . . 147
13.3 Behavior when few available Tasks or Bins . . . 148
14. Appendix . . . 149
14.1 Error Code Overview . . . 149
14.2 Property Overview . . . 152
15. Index . . . 153

1. Read this first

This part of the manual explains the structure and use of this AutoStore manual. Reading the following sub-chapters in detail will help you to get the most out of your AutoStore system.

1.1 About this manual

This Operation Manual contains comprehensive information related to all aspects Interface operation.

The manual is intended for use by category 3 and 4 AutoStore service technicians, who are qualified service personnel approved by AutoStore AS.

> **NOTICE**
> **For a description of the different user categories see the Safety Concepts manual.**

1.2 Conventions and Standards

This section describes the typographical conventions that have been established for AutoStore manuals.

Data Input

Throughout this manual, the following conventions indicate data input and output via the computer:

- Messages displayed on the screen are represented by capitalizing the initial letter of each word and by italicizing each word. For example, "*Close The Robot Door*".
- Statuses and modes are written in capital letters. For example, RUNNING.
- Input that is entered by keyboard is identified by quotation marks: single quotes for single characters, double quotes for strings.
- For brevity, expressions such as “choose File -> Directories” are used rather than “pull down the File menu and choose Directories.”
- Any command is enclosed in angle brackets < > represents a single keystroke. For example, “press <F1>” means press the key labeled F1.
- Any command that requires pressing two or more keys simultaneously is shown with a plus sign connecting the keys. For example, “press <Shift> + <F1>” means press and hold the <Shift> key and then press the <F1> key.
- Any button that you click on the screen or on a physical panel is represented in bold face letters. For example, “click **Close**” or "push **Retry**".

Measurements

Unless specified otherwise:

- All measurements are metric
- Diagrams are read from left to right
- All time references are in UTC

1.3 Contacting us

There are several ways to contact AutoStore:

Assistance

For technical support and ordering information, visit us on the Web:

www.autostoresystem.com

support@autostoresystem.com

Telephone: +47 5276 3660

1.4 Signs and Notices

These WARNING signs were prepared to enhance your personal safety. You should carefully read and follow ALL procedures and recommendations provided through these WARNING, CAUTION, and NOTICE signs.

⚠ WARNING
A WARNING indicates a situation that could result in harm, serious bodily injury, or death if the warning is ignored.

⚠ CAUTION
A CAUTION indicates a situation that could result in damage to the module if the caution is ignored.

✳ NOTICE
A NOTICE indicates that interesting or helpful information is being provided.

1.5 Changes to the Manual

In order to suggest changes to this manual, please forward comments (preferably in English) to:

Editors, Technical Documentation
AutoStore AS
Stokkastrandvegen 85
5578 Nedre Vats
Norway

Alternatively, please send an e-mail message to the Technical Editors at:

documentation@autostoresystem.com

You are encouraged to report errors or omissions in the text or index.

1.6 General Safety Information

In accordance with our commitment to customer service and safety, this module complies with the EEA requirements governing the CE Mark including the Low Voltage Directive.

Designed, manufactured, and tested in an ISO9001 registered facility, this module has been shipped to you from our manufacturing facility in a safe condition.

This module must be used as described in this manual. Any use of this module in a manner other than that described here may result in damage and/or injury.

General Notice on Lifting and Handling of AutoStore modules

> ⚠ CAUTION
> *Always follow local safety guidelines!*

For your safety, and in compliance with international regulations, the physical handling of the modules requires an adequate lifting device for lifting and/or moving. Modules may be too heavy and/or bulky for one person alone to handle safely.

Personal Protective Equipment

- Always use personal protective equipment where applicable

Notice on the Proper Use of AutoStore modules

In compliance with international regulations: if this product is used in a manner not specified by AutoStore, the protection provided by the product could be impaired.

1.7 General Precautions

Observe the following safety precautions when you install, operate, or carry out work on an AutoStore module:

- The system shall be operated by trained personnel only. Read the manuals before starting the system and make sure that you are familiar with the warnings and safety precautions!
- The module can only operate at full capacity if the system is in good condition and properly set up.
- Only use fuses specified by type and current rating. Do not use modified fuses and do not short-circuit the fuse holder.
- For modules where a power cord is supplied, this must be inserted into a power outlet with a protective earth contact (ground). When using an extension cord, make sure that the cord also has an earth contact.
- Do not change the external or internal grounding connections. Tampering with or disconnecting these connections could endanger you and/or damage the system.
- The module is properly grounded in accordance with regulations when shipped.
- Never run the system without the covering mounted. Permanent damage can occur. When leaving the system, make sure that all protective covers and doors are properly connected and closed, and that moving parts are covered or marked in order to provide protection for unqualified personnel!
- Do not turn the module on if you suspect that it has incurred any kind of electrical damage. Instead, disconnect the power cord and contact your distributor for a product evaluation. Do not attempt to use the module until it has been evaluated (electrical damage may have occurred if the system shows visible signs of damage, or has been subjected to severe stress).
- Damage can also result if the module is stored for prolonged periods under adverse conditions (e.g., subjected to heat, water, etc.).
- Unless instructed otherwise, always disconnect the power cord before attempting any type of maintenance.
- Do not place any objects – especially containers with liquids – on the module. Leaking liquids can come into contact with electronic components and cause a short circuit.

2. Introduction

⚠ WARNING
Users must have fulfilled the mandatory AutoStore Interface training course before implementing the AutoStore Interface.

AutoStore provides an Application Programming Interface (API). It enables the customer's warehouse management system (WMS) to integrate with AutoStore.

AutoStore supports two different levels of integration:

- Task Interface
- Bin Interface

The Task Interface operates at a picking list level and offers a higher level of abstraction than the Bin Interface.

The Bin Interface operates at a lower level and gives the WMS more control of the process. It places more responsibility for grouping, preparation, and optimization logic on the WMS.

Each installation must choose to use either the Task Interface or the Bin Interface. The two alternatives are further compared in the next section (see Task Interface and Bin Interface on page 12).

✳ NOTICE
Task Interface and Bin Interface cannot be combined!

The AutoStore Interface includes a Port Interface API that enables the integration of custom port implementation. The Port Interface API is a set of methods that are available via the Task Interface as well as the Bin Interface.

When standard AutoStore Port hardware is used, the Bin exchange process in the Ports is handled by AutoStore and the Port Interface is not needed.

When an installation uses customized Port solutions, such as conveyor belts into the ports or special Port hardware, the Port Interface is required. In these cases, the Port Interface synchronizes the insertion and removal of Bins between the Grid and the Ports.

The Port Interface is described in the Port Interface (on page 117) chapter and applies to both the Task Interface and the Bin Interface.

The AutoStore Log Publisher is a separate module that provides feedback from AutoStore to the WMS. It sends an event stream that includes information about the physical Bin locations and preparation status. This information is required for WMS implementations that use the Bin Interface. WMS implementations that use the Task Interface may use this information for certain optimizations, but this is not required. The Log Publisher Interface is further described in Log Publisher Interface (on page 140) chapter.

2.1 Task Interface and Bin Interface

The Task Interface offers integration at a higher level of abstraction than the Bin Interface. The Task Interface uses concepts like tasks, categories, and shipment.

When you use the Task Interface:

- the WMS maintains a queue of picking lists (task groups)
- AutoStore handles the Bin preparation
- AutoStore plans the jobs for each Port
- the WMS sets the properties of the task groups and Ports to specify priorities and requirements

The Bin Interface offers integration at a lower level and gives the WMS more control of the process.

When you use the Bin Interface:

- the WMS specifies the exact Bin sequence at all Ports
- the WMS handles the Bin grouping
- the WMS plans the jobs for each port

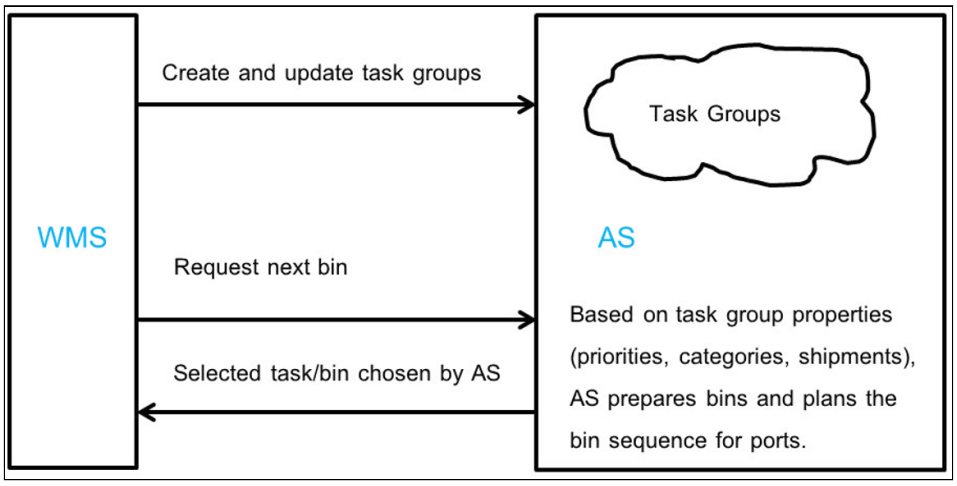
With the Bin Interface, the WMS takes a bigger responsibility for the performance. Many optimizations that are usually provided by the Task Interface must now be handled by the WMS.

2.1.1 Task Interface

The WMS maintains a queue of picking lists (task groups). A task group specifies a collection of Bins (tasks) that should be picked together. A task specifies the delivery of one specific Bin. Tasks are always grouped into a task group structure.

A task group has a set of properties that specify its priority and requirements. It is possible to restrict which Ports may handle the task group (by its category), and multiple task groups can be grouped together (by the shipment property). The sequence and grouping within a task group may be specified by properties on the tasks. The properties of task group and task are further explained in the chapter Basic Concepts (on page 23).

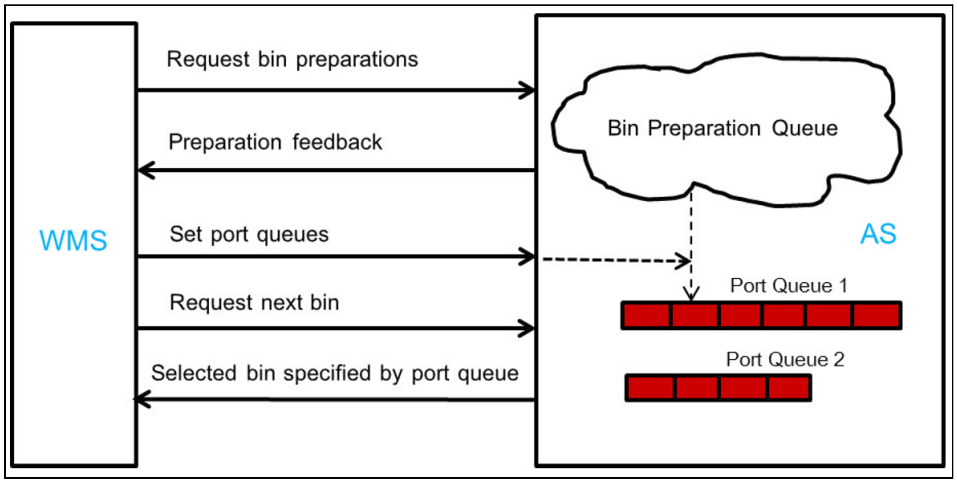
AutoStore decides the task group sequence at the Ports. AutoStore will usually follow the priorities of the task groups. AutoStore will always aim to optimize the total throughput, and may therefore occasionally choose lower priority jobs. If Bins with a high priority are busy or blocked by other jobs, AutoStore will select other task groups in order to avoid waiting time in the Ports.



2.1.2 Bin Interface

The WMS specifies a preparation queue with Bins that should be prepared for use (i.e. placed in a position where they are readily accessible).

For each Port, the WMS specifies a Port queue that defines the Bins that should be delivered to the Port. Therefore, the WMS must handle the grouping of Bins and resolve conflicts when the same Bin is needed at several Ports. The WMS must consider the physical positions and preparation status of the Bins in order to achieve optimum performance.



2.1.3 Comparison

Task Interface:

- Easier to implement. Simple interface (picking list model)
- Covers most standard requirements
- Some advanced/specific requirements can be difficult to implement efficiently
- AutoStore decides the picking sequence

Bin Interface:

- The WMS has full control of the Bin sequence
- Possible to implement advanced requirements/scenarios that are difficult to express with the Task Interface
- More complex to implement
- Requires good understanding of AutoStore behavior, such as preparation/digging/transportation costs
- WMS must plan Bin preparations and Port queues
- More responsibility for the WMS: the WMS implementation and strategies will have big impact on performance/throughput

The Task Interface covers the most usual requirements. Many warehouse processes follow the picking list model, and the properties of the task groups support many common requirements. However, there are scenarios that do not fit the model supported by the Task Interface. Some examples can be production processes or batch picking processes where the Ports combine multiple picking lists.

The Bin Interface is more flexible and gives the WMS full control of the Bin sequence.

> ✳ NOTICE
> **Batch picking processes may still use the Task Interface when the picking lists can be combined into task groups in advance. However, the Bin Interface allows a more dynamic approach since the grouping can be decided later.**

2.2 Technical Interface

This document does not describe the installation and configuration of the AutoStore software modules.

The software architecture and the software modules are described in *AS-50127 - Operation Manual - Software.*

The AutoStore Interface consists of:

- An XML interface via HTTP where the WMS makes method calls to AutoStore.
  The Bin Interface and Task Interface have different specifications of the XML messages.
- A stream interface via TCP where AutoStore sends information to the WMS.
  This stream of information is provided by the Log Publisher interface.

2.2.1 XML Message Format

Each message must:

- be well-formed XML
- begin with an XML declaration that specifies
  - the XML version (1.0)
  - the encoding (optional, `iso-8859-1` and `utf-8` are supported)

The message format is loosely based on XML-RPC. It supports named parameters with types that are not defined globally but rather on a per method basis. Naming conventions also differ from XML-PRC.

2.2.1.1 Request

The root element of the method request is `<methodcall>`, and the name of the method is given as a string in the first level child element `<name>`. The method parameters are given in the first level child element `<params>`. The method parameters are specific to each method.

XML request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>add_task</name>
   <params>
      <taskgroup_id>123004</taskgroup_id>
      <tasks>
         <task>
            <task_id>420233</task_id>
            <bin_id>23255</bin_id>
         </task>
      </tasks>
   </params>
</methodcall>
```

2.2.1.2 Response

The root element of the returned XML document is always `<response>`.

If the method call is successful, a child element named `<params>` will be created. It contains the returned data. If no data is returned, `<params>` will be empty.

XML response example:

```
<?xml version="1.0"?>
<response>
   <params/>
</response>
```

If the method call is not successful a child element named `<fault>` will be created . It will contain another child element named `<code>` that contains the numeric error code.

XML fault response example:

```
<?xml version="1.0"?>
<response>
   <fault>
      <code>1009</code>
   </fault>
</response>
```

2.2.2 Using the AutoStore Interface

Messages are sent as HTTP POST requests to one of the following URLs:

Interface	URL
Task Interface	http://<server>/AsInterfaceHttp/AutoStoreHttpInterface.aspx
Bin Interface	http://<server>/AsInterfaceHttp/BinInterface.aspx

If the default virtual directory (AsInterfaceHttp) was overridden during installation, then that part of the URL will change as well.

⚠ WARNING
Do not use the WSDL file. We do not support the SOAP interface. It will be removed in future releases. You shall only use HTTP POST method enclosing the XML messages that are specified in the Interface Specification.

⚠ WARNING
Never use Task Interface and Bin Interface in the same implementation!

2.3 Feature Compatibility

The functionality and API specifications in this manual applies to the October 2022 software release.

This manual also describes some features and functionality that is planned for future software releases.

Integration and simulation of future features and functionality are supported by release candidates of the corresponding software releases and WMS SDKs.

2.3.1 Features that require AS Router

The following features and modules are not supported by AS Planner:

- Strict sequence of task groups (port mode = SEQUENCE)
- Fusion Port
- PickUpPort
- Configurable look-ahead visibility (getnexttaskgroups)
- Frozen Zone and Multi Temperature Grid installations
- AutoFlush in port

2.3.2 Software Release October 2022

The following features was changed in the October 2022 software release:

- Controlling Bin Location supported by AS Router.
  - The features that are marked as deprecated are not supported by AS Router.
  - AS Router only supports one code per cell. Use ranges when multiple codes are needed.
- The method flushportbin can also be used in open ports.

2.3.3 Software Release April 2023

The following features will be changed in the April 2023 software release:

- Add support for PickUpPort
- Add support for controlling strict sequence of task groups (port mode = SEQUENCE)
- The method *getportweight*
- Support Bin ID up to 4.000.000

2.3.4 Software Release October 2023

The following features will be changed in the October 2023 software release:

- Add support for Fusion Port
- Add support for MultiPort configuration

3. Basic Concepts

This section describes basic concepts that are common to both interfaces (Task Interface and Bin Interface).

3.1 Bin

The Bins are the storage locations inside the AutoStore system.

Every Bin has a unique identifier that is assigned by the AutoStore system. The WMS should use this identifier to identify storage locations.

The WMS must keep track of the inventory of each Bin. AutoStore does not maintain detailed information about the Bin content, such as products and quantities.

Every Bin has an actual state that includes its physical location, preparation state, and mode in port. The WMS may receive the current state through the *Log Publisher Interface*.

3.1.1 Bin Properties

bin_id:

The bin_id is the unique identifier of each Bin.

content:

The content code assigned to the bin. The content code is updated by the WMS.

bin_type:

The type of the Bin as specified by the following codes:

bin_type	Description
1	Standard Bin. Height 330 mm
2	Standard Bin. Height 220 mm
5	Standard Bin. Height 425 mm

3.1.2 State

Physical Location:

The current physical location of a bin includes xpos, ypos, depth, and grid_id. If the bin is being moved by a robot, then only the grid_id is set.

Bin Mode:

The current bin_mode reflects both the preparation status and if the bin is open in a port. The bin mode can be one of the following:

Mode	Description

GRID	Bin is in a Grid.
FORECAST	Bin is in a Grid and is forecast.
PREPARED	Bin is prepared in a Grid.
PORT	Bin is in a Port, ready to be opened.
TOGRID	Same as PORT, but the bin is requested back to Grid because it is needed elsewhere.
OPEN	Bin is open in a Port.
CLOSED	Bin is in a Port, ready to be returned to Grid.
OUTSIDE	Bin is removed (temporary) from the system.

Normally, a bin changes mode from **GRID** to **PREPARED** to **PORT** to **OPEN** to **CLOSED** and back to **GRID**.

3.1.3 Bin Content Code

Every bin is assigned a content code. This property, an integer between 1 and 32767, is assigned and maintained by the WMS.

The WMS may define its own set of content codes. A standard convention is to build the content code of the number of compartments in the bin together with the number of used compartments. If the installation uses different bin types (e.g. ESD and non ESD), then the content code would normally also reflect the bin type.

The content code may be used to let AutoStore select an actual bin. This is intended to be used in situations where many bins are equal from the WMS's point of view and where many of these bins also are normally available without preparation. Typically, this is used during the putaway process when the WMS just wants an empty bin.

3.1.3.1 Content Code Example

The content code can be used to indicate whether a Bin is empty or not. Conventionally, the code represents the number of compartments inside the Bin as well as the number of compartments with goods inside.

See the table below for examples:

Content Code	Bin Division	Bin Usage
10	Undivided Bin	Empty Bin
11	Undivided Bin	Goods in Bin
20	Divided into 2 compartments	Both compartments are empty
21	Divided into 2 compartments	Goods in one compartments
22	Divided into 2 compartments	Goods in both compartments
40	Divided into 4 compartments	All compartments are empty
41	Divided into 4 compartments	Goods in one compartments
42	Divided into 4 compartments	Goods in two compartments
43	Divided into 4 compartments	Goods in three compartments

44	Divided into 4 compartments	Goods in all compartments

NOTICE

The WMS can define its own set of content codes and semantics.

3.1.4 Automatic Preparation of Empty Bins

AutoStore provides automatic preparation of empty Bins. This background process operates with both the Task Interface and the Bin Interface.

The purpose of this preparation process is to ensure that a number of empty Bins are available (prepared) for the putaway process. When the WMS uses content codes to control the putaway process, then the WMS does not explicitly request preparation of these Bins in advance.

AutoStore will by default prepare Bins with content code 0. If the WMS wants to use a different content code, or multiple content codes, this can be configured by using the AutoStore Console. In the case of multiple content codes it is possible to configure the relative distribution between them.

It is possible for the WMS to control the putaway process without using content codes. The WMS can handle the putaway process in a similar way to the picking process. In such cases, the WMS will use the standard preparation queue to ensure that there are empty Bins available (or if the Task Interface is used, the WMS will create task groups for the putaway process). In cases where the WMS controls the putaway process without using content codes, the automatic preparation of empty Bins should be disabled.

During the initial ramp-up phase (initial inventory insertion phase) more empty bins are consumed. Therefore, it is recommended to configure a higher value for empty bins preparation during this phase, and restore to normal value after this phase.

3.1.4.1 Configurations

Empty Bin Content

This configuration allows a list of content codes to be set that AutoStore should consider empty and prepare for the putaway process.

Empty Bins Factor

This configuration sets the ratio between the content codes used for empty Bins. A factor is specified for each content code.

Planner Prepare Empty Bins

This configuration sets how many of the storage cells in the grid should be used for empty bin preparation. Changing this value could impact the putaway performance versus the picking performance.

3.2 Port

A Port is the operator workstation where Bins are delivered, removed, or inserted.

One or more grid cells belong to the Port and are used when Robots delivers Bins to and gets Bins from the Port.

3.2.1 Port Properties

The following properties are common for both Task Interface and Bin Interface. (Properties that are interface dependant are described in the respective chapters.)

port_id:

The port_id is a unique identifier that identifies a port.

port_mode:

The current mode of operation.

The port_mode is set by the *openport* method.

The port_mode is set to **CLOSED** by the *closeport* method.

selected_bin:

The selected_bin is the next Bin to be handled in the Port. It can be the current open bin in the Port or on the way to be delivered in the Port.

The selected_bin property is set to the bin_id returned by the *openbin* method.

The selected_bin property is cleared by the *closebin*, *removebin*, and *closeport* methods.

In port mode **AUTO**, the selected_bin is cleared when the bin is confirmed in port.

The selected_bin will be delived to port and is not affected by changes in the port queue or task groups. The only way to cancel a selected_bin before it is delivered is to close the port.

3.2.2 Port Mode

The port_mode is the current mode of operation of the port. The port_mode is controlled by the WMS.

Ports that are not in use by operators should be closed, set to Port Mode CLOSED. AutoStore will plan work and allocate robot resources for open ports, so in order to avoid wasted performance it is important to close ports that are not in active use. Also, there may be bins in an open port that are needed by other ports, and these bins will be blocked until the bins are handled (or the port is closed).

The following modes are available:

Mode	Description
CLOSED	The port is closed.
OPEN	Default mode. The port is open for picking or receiving goods.
AUTO	The port is open for picking or receiving goods, and will automatically open bins. (Deprecated.)
INSERT	Special mode. The port is open for bin insertion.
SWAP	Special mode. The port can be used for exchanging bins between grids.
MOVE	Not used.
SEQUENCE	Special mode. Task Interface only. The port is open and will deliver the defined sequence of task groups.
DISABLED	The port is disabled. See section about MultiPort.
MULTIPORT_PUPPET	The port ID is disabled. The port module is controlled by a MultiPort.

The port mode **INSERT** is used to add bins to AutoStore. This mode is used during the installation of of the system. This mode may also be used in installations where the bins temporarily leave the grid (for instance where multiple ports are connected through conveyor-belt systems).

The port mode **AUTO** automatically opens bins in the Port, and is used in conjunction with the Port Interface. In this mode, the WMS does not use the *openbin* and *closebin* messages. AutoStore will automatically generate *openbin* and *removebin* commands and provide a stream of bins according to the port queue. The port mode AUTO is deprecated and should not be used by new installations.

3.3 Grid

Connected grids in an installation have common x and y coordinates, but the depth of a bin is specified by the local offset from the top of the grid.

In the current version, the `grid_id` property has no function. From Log Publisher the value is always 1.

4. Task Interface

The Task Interface provides an interface that is based on a picking list model.

4.1 Basic Concepts

This section describes the basic terms and objects of the Task Interface.

4.1.1 Task Group

A task group is a collection of tasks that are picked together.

4.1.1.1 Task Group Properties

taskgroup_id:

The taskgroup_id is the unique identifier for each task group.

The task group identifier cannot be reused until 24 hours after the task group was deleted or completed.

shipment:

A shipment is a collection of task groups that are normally picked together.

The shipment is a numerical value that identifies a shipment. Task groups with an equal shipment property belong to the same shipment.

The shipment property is a single value; therefore a task group cannot belong to multiple shipments.

A task group is not required to belong to a shipment.

category:

The category property restricts the picking of a task group to Ports which are allowed to pick tasks from a given category.

A task group must belong to one single category.

req_time:

The req_time is the requested time of the task group, and is used to prioritize different task groups.

It is not logically connected to the current system time, but is a value that is used for sorting (together with the property priority).

Whether a task group can be started or finished to be picked at a Port until req_time, depends on the available system resources and the operational situation (e.g. composition and status of the task queue).

priority:

The task groups are prioritized by their requested time and priority. Task groups with earliest requested time have the highest priority. Task groups with equal requested time are ordered by their priority (lowest value first; priority 1 before priority 2).

AutoStore does not follow this priority order strictly. AutoStore may deliver lower prioritized tasks in order to achieve a higher throughput.

start_time:

If the start_time property is set, the task group will not be selected by AutoStore until the specified start time is reached.

The start_time property is optional; by default task groups can be selected by AutoStore immediately after they are created.

suspended:

A task group that is suspended will not be selected by AutoStore.

The suspended property can be set by the *closeport* method if the WMS closes the port while the task group is selected.

The suspended property is cleared when the WMS explicitly selects the task group.

4.1.2 Task

A task specifies the retrieval of one specific Bin. Tasks are always grouped into a task group structure which defines a set of one or more tasks that must be handled together.

4.1.2.1 Properties

task_id:

The task_id is a unique identifier for each task.

This identifier must be globally unique among all task groups.

The task_id identifier cannot be reused until 24 hours after its task group was deleted or completed.

bin_id:

The bin_id identifies the Bin that should be retrieved.

item:

Tasks with equal item property will be grouped and picked together.

The item property is optional.

order:

Within a task group, tasks with lower order will be picked before tasks with higher order.

The order property is optional.

Tasks that are grouped with equal item property are not allowed to have conflicting order property.

Tasks that are not assigned an order may be picked in any order, even between tasks with the same order.

estpicktime:

Each task can be assigned an expected handling time (seconds).

This information is used by AutoStore in order to optimize performance.

The estpicktime property is optional.

4.1.3 Port Properties

4.1.3.1 Select Requirements

The Ports' select requirements are the properties (req_time and select) that restrict which bins AutoStore will select. The select requements are specified as parameters to the *openport* method when a new port session is started.

The select requirements also affect the AutoStore Bin preparation process.

req_time:

The req_time property is an optional filter that restricts which task groups AutoStore will deliver to the Port. AutoStore will only select task groups with req_time equal to or earlier than this property.

This property can be used to ensure that AutoStore only delivers high priority task groups to the port, since it avoids delivery of task groups with req_time after the specified limit (port req_time).

select:

The select property is a collection of categories or a collection of Bin content codes. It controls which task groups and Bins AutoStore will select to the Port.

AutoStore will only select task groups that belong to the specified categories, or Bins that have the specified Bin content.

The maximum number of selected content codes or categories is 32.

4.1.3.2 Selected Bin

When a bin is selected, multiple related state properties can be set.

selected_bin:

The selected_bin property is set to the bin_id returned by the *openbin* method.

The selected_bin property is cleared by the *closebin, removebin*, and *closeport* methods.

selected_task:

The selected_task property is set to the task_id returned by the *openbin* method.

The selected_task property is cleared by the *closebin, removebin*, and *closeport* methods.

selected_taskgroup:

The selected_taskgroup property is set to the task group to which the selected task belongs when selected_task is set.

The selected_taskgroup property is cleared by the *closebin* and *removebin* methods if there are no remaining tasks in the selected task group.

The selected_taskgroup property is cleared by the *closeport* method.

selected_shipment:

The selected_shipment property is set to the shipment to which the selected task group belongs when selected_taskgroup is set.

The selected_shipment property is cleared by the *closebin* and *removebin* methods (in some implementations it is not cleared until to the next *openbin*) if there are no remaining tasks in the selected task group, and there are no remaining unselected task groups in the selected shipment that fulfills the select requirements that were specified by the *openport* method.

The selected_shipment property is cleared by the *closeport* method.

4.2 Balancing Categories

Categories filters which ports that can handle which task groups. Categories do not affect the bin preparation. AutoStore prepares bins based on req_time/priority.

The WMS must ensure that the req_time/priority corresponds with what is actually going to be picked. High priority task groups of categories that are not being used will have a negative impact on performance, because there are limitied space on top of the grid for prepared bins (and also wasted resources spent on preparing bins that are not needed).

4.3 Performance Guidelines

To be able to prepare the bins in time to avoid waiting in ports, the WMS should provide task groups for at least the next 30 minutes of operation. AutoStore needs tasks in advance in order to prepare and plan the bin delivery in time. AutoStore also need a certain queue of work to have sufficient flexibility to keep robots and ports busy. Similarly, tasks should be created at least 30 minutes before they are needed in port. Urgent tasks are possible, but may impact the planned jobs. Also, as described earlier, the queue of tasks must be balanced wrt. categories. Hence, the 30 minute rules also applies for each category.

To optimize performance, it is recommended to minimize the restrictions. Restrictions such as item grouping, sequence requirements, shipment grouping and category requirements reduces the flexibility. These features should only be used when they are required by the warehouse processes.

4.4 Port Methods

4.4.1 Open Port

The **openport** method establishes that a Port is ready for work.

The message specifies what kind of Bins or tasks AutoStore should select for that Port.

4.4.1.1 Parameters

port_id: (mandatory)

Identifies the Port.

select: (optional)

A collection of either categories or Bin content codes. The collection cannot contain categories as well as Bin content codes.

Sets the Port's select property.

If this parameter is omitted (no categories and content codes are specified), the Port is open for Bin insertion (**INSERT** mode).

req_time: (optional)

Sets the Port's req_time property.

port_mode: (optional)

Specifies a special mode of operation. Valid Port modes are **INSERT**, **OPEN** and **SWAP**.

Both modes **OPEN** and **SWAP** requires that the select parameter has been specified.

If the port_mode parameter is omitted, the Port is opened in mode **OPEN** if the select parameter has been specified, or in mode **INSERT** if the select parameter has been omitted.

4.4.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "openport" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element req_time { xsd:dateTime } ?,
      element select {
         element category { xsd:short { minInclusive = "1" } } +
```

```
            | element content { xsd:short { minInclusive = "1" } } +
         } ?,
         element port_mode { "INSERT" | "OPEN" | "SWAP" } ?
      }
   }
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>openport</name>
   <params>
      <port_id>4</port_id>
      <req_time>2008-12-12 14:00</req_time>
      <select>
         <category>1</category>
         <category>2</category>
      </select>
   </params>
</methodcall>
```

4.4.1.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1002	The specified Port is already open.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1007	Parameter category is not valid (invalid format or out of range).
1008	Parameter req_time is not valid (invalid format or out of range).
1048	System is not available.
1050	Too many categories or content codes.

4.4.2 Open Bin

The **openbin** method requests a Bin in the Port.

AutoStore selects a Bin or task according to the the Port's current selection and the Port's select requirements. AutoStore will not select a task group or a shipment that is already selected by another Port.

The WMS may explicitly select a specific Bin, task, task group, or shipment.

The WMS cannot explicitly select anything that conflicts with the Port's selected Bin or selected task (a previous openbin call that is not yet closed).

The WMS cannot explicitly select a task, a task group, or a shipment that conflicts with the Port's selected task group or selected shipment.

The WMS cannot explicitly select a task group (or a task from a task group) that is already selected by another Port. The WMS may select a shipment (or a task or task group from a shipment) that is already selected by another Port.

The WMS may select a specific Bin (ad-hoc bin) that conflicts with the selected task group or selected shipment. The Port's selected task group and selected shipment will remain selected, and will continue after the ad-hoc Bin is closed.

An explicit select of a task may conflict with, and will override, the task's order and item properties.

An explicit select may conflict with, and will override, the Port's select requirement.

It is possible to explicitly select a task or task group that conflicts with the Port's select requirements, for instance of a different category. The WMS only needs to override in the first openbin call in a task group. The task group will be selected and AutoStore will select the rest of the task group even if it conflicted with the Port's select requirements. After the task group is complete, AutoStore will select a new task group according to the Port's select requirement.

Selecting a specific shipment however, will not override the Port's select requirements. AutoStore will select the specified shipment, but will select a task group within that shipment that fulfills the Port's select requirements.

A task group cannot be selected by more than one Port at the same time. A shipment may be selected by more than one Port at the same time if, and only if, the WMS explicitly selects it for another Port.

In order to optimize performance, AutoStore should select the Bin or task. Explicit selections will inflict a performance penalty, especially if the selection conflicts with the Port's select requirement.

4.4.2.1 Parameters

port_id: (mandatory)

Identifies the port.

bin_id: (optional)

Explicitly selects a specific bin.

task_id: (optional)

Explicitly selects a specific task.

taskgroup_id: (optional)

Explicitly selects a specific task group.

shipment: (optional)

Explicitly selects a specific shipment.

4.4.2.2 Response

port_id:

Identifies the Port.

bin_id:

The Port's selected Bin.

task_id:

The Port's selected task.

4.4.2.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "openbin" },
   element params {
```

```
            element port_id { xsd:short { minInclusive = "1" } },
            ( element bin_id { xsd:int { minInclusive = "1" } }
            | element task_id { xsd:int { minInclusive = "1" } }
            | element taskgroup_id { xsd:int { minInclusive = "1" } }
            | element shipment { xsd:int { minInclusive = "1" } }
            ) ?
         }
      }
```

Compact syntax (Response):

```
element response {
   element params {
   element port_id { xsd:short { minInclusive = "1" } },
   element bin_id { xsd:int { minInclusive = "1" } },
   element task_id { xsd:int { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>openbin</name>
   <params>
      <port_id>4</port_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <port_id>4</port_id>
      <bin_id>14861</bin_id>
      <task_id>310022</task_id>
   </params>
</response>
```

4.4.2.4 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1012	Parameter task_id is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1014	Parameter shipment is not valid (invalid format or out of range).
1015	There are no tasks or Bins that fulfill the Port select requirements.

1016	There are no Bins ready that fulfill the Port select requirements.
1017	The specified task does not exist.
1018	The specified task group does not exist.
1019	The specified task group is selected by another Port.
1020	The openbin conflicts with a previous openbin call that is not yet closed.
1021	The openbin conflicts with the Port's selected shipment.
1022	The openbin conflicts with the Port's selected task group.
1023	Multiple selection parameters are not allowed.
1048	System is not available.

4.4.3 Close Bin

The **closebin** method closes the Bin in the Port.

The Bin must be opened and ready in the Port before the Bin can be closed. The WMS must specify the correct task and Bin when it closes a Bin.

The WMS may update the content of the Bin.

The task is completed by the *closebin* method, unless overridden by the *taskcomplete* parameter.

4.4.3.1 Parameters

port_id: (mandatory)

Identifies the Port.

bin_id: (mandatory)

Must be equal to the open Bin.

task_id: (optional)

If the open Bin belongs to an open task, this parameter must be equal to the open task.

content: (optional)

Sets the content of the Bin. If this parameter is not provided, the content of the Bin is not changed.

taskcomplete: (optional)

Specifies if the open task should be completed. Default value is "true".

4.4.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "closebin" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } },
      element task_id { xsd:int { minInclusive = "1" } } ?,
      element taskcomplete { xsd:boolean } ?,
      element content { xsd:short { minInclusive = "1" } } ?
   }
```

```
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>closebin</name>
   <params>
      <port_id>4</port_id>
      <bin_id>14861</bin_id>
      <task_id>310022</task_id>
      <content>10</content>
   </params>
</methodcall>
```

4.4.3.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1012	Parameter task_id is not valid (invalid format or out of range).
1024	Mandatory parameter bin_id is missing.
1025	Parameter taskcomplete is not valid (invalid format or out of range).
1026	The Port does not have a selected Bin.
1027	The Bin is not yet ready in the Port.
1028	The Bin and task parameters conflict with the open Bin and task.
1048	System is not available.

4.4.4 Close Port

The **closeport** method closes the Port.

If there is an open Bin in the Port, the Bin is closed. The open task is not completed, and the Bin's content is not updated.

The Port's select requirements are cleared by the *closeport* method.

The Port's selected shipment, selected task group, selected task and selected Bin are cleared by the *closeport* method.

4.4.4.1 Parameters

port_id: (mandatory)

Identifies the Port.

suspend_taskgroup: (optional)

Sets the task group's suspended property. Default value is "false".

The task group is not bound to the Port after the Port is closed. To prevent an unfinished task group from being processed at a different Port or by a different user, the WMS may suspend the task group. The task group is suspended until a Port selects the task group by using *openbin* that explicitly specifies that task group. (The suspended property may also be updated by using *update_taskgroup*).

4.4.4.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "closeport" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element suspend_taskgroup { xsd:boolean } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>closeport</name>
   <params>
      <port_id>4</port_id>
      <suspend_taskgroup>1</suspend_taskgroup>
   </params>
</methodcall>
```

4.4.4.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1048	System is not available.

4.4.5 Insert Bin

The **insertbin** method establishes that a Bin has been inserted in a Port and closes the inserted Bin.

The Port cannot have a selected Bin.

The WMS may update the Bin's content.

4.4.5.1 Parameters

port_id: (mandatory)

Identifies the Port.

bin_id: (mandatory)

Identifies the Bin.

content: (optional, but mandatory when bin_type is set)

Sets the Bin's content.

If this parameter is omitted, the Bin will keep its previous content. If the Bin has never been in the system before, the content is set to "0".

bin_type: (optional, but mandatory if the bin is not defined yet)

Specifies the Bin's type.

This parameter is mandatory if the Bin is not defined (the first time the Bin is inserted). In installations where Bins are removed and inserted, the bin_type parameter is not required in subsequent inserts.

4.4.5.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "insertbin" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } },
      element content { xsd:short { minInclusive = "1" } } ?,
      element bin_type { xsd:short { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>insertbin</name>
   <params>
      <port_id>4</port_id>
      <bin_id>16477</bin_id>
      <content>10</content>
   </params>
</methodcall>
```

4.4.5.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1029	The specified Bin is already inserted.

1030	The Port has a selected Bin.
1048	System is not available.
1058	Bin is still outside.
1073	Parameter bin_type is not valid (invalid format or out of range).
1077	Maximum number of Bins already inserted.
1083	No room for given content.

4.4.6 Remove Bin

The **removebin** method establishes that the open Bin has been removed from a Port.

The specified Bin must be open and ready in the Port.

4.4.6.1 Parameters

port_id: (mandatory)

Identifies the Port.

bin_id: (mandatory)

Must be equal to the open Bin in the Port.

task_id: (optional)

If the open Bin belongs to an open task, this parameter must be equal to the open task.

taskcomplete: (optional)

Specifies if the open task should be completed. Default value is "true".

4.4.6.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "removebin" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } },
      element task_id { xsd:int { minInclusive = "1" } } ?,
      element taskcomplete { xsd:boolean } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>removebin</name>
   <params>
      <port_id>4</port_id>
      <bin_id>14861</bin_id>
      <task_id>310022</task_id>
   </params>
</methodcall>
```

4.4.6.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1012	Parameter task_id is not valid (invalid format or out of range).
1017	The specified task does not exist.
1024	Mandatory parameter bin_id is missing.
1025	Parameter taskcomplete is not valid (invalid format or out of range).
1026	The Port does not have a selected Bin.
1027	The Bin is not yet ready in the Port.
1028	The Bin and task parameters conflict with the open Bin and task.
1048	System is not available.

4.4.7 Update Bin

The **updatebin** method updates the content of a Bin.

4.4.7.1 Parameters

bin_id: (mandatory)

Identifies the Bin that should be updated.

content: (mandatory)

Sets the Bin's content.

4.4.7.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "updatebin" },
   element params {
      element bin_id { xsd:int { minInclusive = "1" } },
      element content { xsd:short { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>updatebin</name>
   <params>
      <bin_id>14861</bin_id>
      <content>11</content>
   </params>
</methodcall>
```

4.4.7.3 Error Codes

Error Code	Description
1006	Parameter content is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1024	Mandatory parameter bin_id is missing.

4.4.8 Get Port Status

The **getportstatus** message retrieves information about the current status of the Port with the specified Port number.

4.4.8.1 Parameters

port_id: (mandatory)

Specifies the Port.

4.4.8.2 Response

port_id:

Identifies the Port.

mode:

The current mode of operation of the Port.

selected_bin:

Which Bin is currently selected at the specified Port (0 if no Bin is selected).

selected_task:

Which Task is currently selected at the specified Port (0 if no Task is selected).

is_ready:

Tells if the Bin is ready to be handled in the Port.

4.4.8.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getportstatus" },
   element params {
      element port_id { xsd:int }
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element port {
         element port_id { xsd:int },
         element mode { xsd:string },
         element selected_bin { xsd:int },
         element selected_task { xsd:int },
         element is_ready { xsd:boolean }
      }
   }
}
```

Request example:

```
<?xml version="1.0" encoding="utf-8"?>
<methodcall>
   <name>getportstatus</name>
   <params>
      <port_id>1</port_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0" encoding="utf-8"?>
<response>
   <params>
      <port>
         <port_id>1</port_id>
         <mode>OPEN</mode>
         <selected_bin>34401</selected_bin>
         <selected_task>1190504</selected_task>
         <id_ready>0</is_ready>
      </port>
   </params>
</response>
```

4.4.8.4 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).

4.5 Task Methods

4.5.1 Create Taskgroup

The **createtaskgroup** method creates a new task group, including the collection of tasks that belong to the task group.

A task groups must have a unique taskgroup_id.

4.5.1.1 Parameters

taskgroup_id: (mandatory)

Sets the taskgroup_id property.

shipment: (optional)

Sets the shipment property.

category: (mandatory)

Sets the category property.

req_time: (mandatory)

Sets the req_time property.

priority: (mandatory)

Sets the priority property.

start_time: (optional)

Sets the start_time property.

suspended: (optional)

Sets the suspended property. The default value is "false".

tasks: (mandatory)

A collection of tasks that belong to the task group.

task_id: (mandatory)

Sets the task_id property of a task.

bin_id: (mandatory)

Sets the bin_id property of a task.

item: (optional)

Sets the item property of a task.

order: (optional)

Sets the order property of a task.

estpicktime: (optional)

Estimated handling time, specified in seconds.

4.5.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "create_taskgroup" },
```

```
      element params {
         element taskgroup_id { xsd:int { minInclusive = "1" } },
         element shipment { xsd:int { minInclusive = "1" } } ?,
         element category { xsd:short { minInclusive = "1" } },
         element req_time { xsd:dateTime } ,
         element priority { xsd:short { minInclusive = "1" } },
         element start_time { xsd:dateTime } ?,
         element suspended { xsd:boolean } ?,
         element tasks {
            element task {
               element task_id { xsd:int { minInclusive = "1" } },
               element bin_id { xsd:int { minInclusive = "1" } },
               element item { xsd:int { minInclusive = "1" } } ?,
               element order { xsd:int { minInclusive = "1" } } ?,
               element estpicktime { xsd:int { minInclusive = "1" } } ?
            } +
         }
      }
   }
```

Request example:

```
   <?xml version="1.0"?>
   <methodcall>
      <name>create_taskgroup</name>
      <params>
         <taskgroup_id>123004</taskgroup_id>
         <shipment>20961</shipment>
         <category>1</category>
         <req_time>2009-03-20 14:00</req_time>
         <priority>20</priority>
         <tasks>
            <task>
               <task_id>420192</task_id>
               <bin_id>10304</bin_id>
               <item>1</item>
               <estpicktime>20</estpicktime>
            </task>
            <task>
               <task_id>420193</task_id>
               <bin_id>33570</bin_id>
               <item>1</item>
               <estpicktime>20</estpicktime>
            </task>
            <task>
               <task_id>420194</task_id>
               <bin_id>22655</bin_id>
               <item>2</item>
               <estpicktime>20</estpicktime>
            </task>
         </tasks>
      </params>
   </methodcall>
```

4.5.1.3 Error Codes

Error Code	Description

1007	Parameter category is not valid (invalid format or out of range).
1008	Parameter req_time is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1012	Parameter task_id is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1014	Parameter shipment is not valid (invalid format or out of range).
1024	Mandatory parameter bin_id is missing.
1025	Parameter taskcomplete is not valid (invalid format or out of range).
1031	Mandatory parameter taskgroup_id is missing.
1032	The task group already exists.
1033	The task already exists.
1034	Mandatory parameter category is missing.
1035	Mandatory parameter req_time is missing.
1036	Mandatory parameter priority is missing.
1037	Parameter priority is not valid (invalid format or out of range).
1038	Parameter start_time is not valid (invalid format or out of range).
1039	Parameter suspended is not valid (invalid format or out of range).
1040	Mandatory parameter task_id is missing.
1041	Parameter item is not valid (invalid format or out of range).
1042	Parameter order is not valid (invalid format or out of range).
1043	The task group has no tasks.
1057	Parameter estpicktime is not valid (invalid format or out of range).

4.5.2 Update Taskgroup

The **update_taskgroup** method updates an existing task group.

It does not update tasks that belong to the task group.

4.5.2.1 Parameters

taskgroup_id: (mandatory)

Identifies the task group that should be updated.

shipment: (optional)

Sets the shipment property.

category: (mandatory)

Sets the category property.

req_time: (mandatory)

Sets the req_time property.

priority: (mandatory)

Sets the priority property.

start_time: (optional)

Sets the start_time property.

suspended: (optional)

Sets the suspended property. The default value is false.

4.5.2.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "update_taskgroup" },
   element params {
      element taskgroup_id { xsd:int { minInclusive = "1" } },
      element shipment { xsd:int { minInclusive = "1" } } ?,
      element category { xsd:short { minInclusive = "1" } },
      element req_time { xsd:dateTime },
      element priority { xsd:short { minInclusive = "1" } },
      element start_time { xsd:dateTime } ?,
      element suspended { xsd:boolean } ?,
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>update_taskgroup</name>
   <params>
      <taskgroup_id>123004</taskgroup_id>
      <shipment>20961</shipment>
      <category>1</category>
      <req_time>2009-03-20 14:00</req_time>
      <priority>20</priority>
   </params>
</methodcall>
```

4.5.2.3 Error Codes

Error Code	Description
1007	Parameter category is not valid (invalid format or out of range).
1008	Parameter req_time is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1014	Parameter shipment is not valid (invalid format or out of range).
1018	The specified task group does not exist.
1031	Mandatory parameter taskgroup_id is missing.
1034	Mandatory parameter category is missing.

1035	Mandatory parameter req_time is missing.
1036	Mandatory parameter priority is missing.
1037	Parameter priority is not valid (invalid format or out of range).
1038	Parameter start_time is not valid (invalid format or out of range).
1039	Parameter suspended is not valid (invalid format or out of range).
1044	The task group is selected by a Port.

4.5.3 Cancel Taskgroup

The **cancel_taskgroup** method cancels an existing task group.

Do not cancel a task group that is selected to a Port.

4.5.3.1 Parameters

taskgroup_id: (mandatory)

Identifies a task group that should be canceled.

4.5.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "cancel_taskgroup" },
   element params {
      element taskgroup_id { xsd:int { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>cancel_taskgroup</name>
   <params>
      <taskgroup_id>123004</taskgroup_id>
   </params>
</methodcall>
```

4.5.3.3 Error Codes

Error Code	Description
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1018	The specified task group does not exist.
1031	Mandatory parameter taskgroup_id is missing.
1044	The task group is selected by a Port.

4.5.4 Add Task

The **add_task** method inserts a task into an existing task group.

4.5.4.1 Parameters

taskgroup_id: (mandatory)

Identifies the task group to which new tasks should be added.

tasks: (mandatory)

A collection of tasks to be added to the task group.

task_id: (mandatory)

Sets the task_id property of a task.

bin_id: (mandatory)

Sets the bin_id property of a task.

item: (optional)

Sets the item property of a task.

order: (optional)

Sets the order property of a task.

estpicktime: (optional)

Estimated handling time, specified in seconds.

4.5.4.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "add_task" },
   element params {
      element taskgroup_id { xsd:int { minInclusive = "1" } },
      element task {
         element task_id { xsd:int { minInclusive = "1" } },
         element bin_id { xsd:int { minInclusive = "1" } },
         element item { xsd:int { minInclusive = "1" } } ?,
         element order { xsd:int { minInclusive = "1" } } ?,
         element estpicktime { xsd:int { minInclusive = "1" } } ?
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>add_task</name>
   <params>
      <taskgroup_id>123004</taskgroup_id>
      <task>
         <task_id>420233</task_id>
         <bin_id>23255</bin_id>
         <estpicktime>20</estpicktime>
      <task>
```

```
   </params>
</methodcall>
```

4.5.4.3 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1012	Parameter task_id is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1018	The specified task group does not exist.
1024	Mandatory parameter bin_id is missing.
1031	Mandatory parameter taskgroup_id is missing.
1033	The task already exists.
1040	Mandatory parameter task_id is missing.
1041	Parameter item is not valid (invalid format or out of range).
1042	Parameter order is not valid (invalid format or out of range).
1057	Parameter estpicktime is not valid (invalid format or out of range).

4.5.5 Cancel Task

The **cancel_task** method cancels a single task.

A task that belongs to a selected task group may be canceled. A task that is selected cannot be canceled.

4.5.5.1 Parameters

task_id: (mandatory)

Identifies a task that should be canceled.

4.5.5.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "cancel_task" },
   element params {
      element task_id { xsd:int { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>cancel_task</name>
```

```
    <params>
       <task_id>123004</task_id>
    </params>
</methodcall>
```

4.5.5.3 Error Codes

Error Code	Description
1012	Parameter task_id is not valid (invalid format or out of range).
1017	The specified task does not exist.
1040	Mandatory parameter task_id is missing.
1045	The task is selected by a Port.

4.6 Query Methods

4.6.1 Get Bin Information

The **getbininfo** method retrieves information about the Bins in the system.

4.6.1.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only information about that specific Bin is returned.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only Bins with bin_id greater than or equal to min_bin_id are returned.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only Bins with bin_id less than or equal to max_bin_id are returned.

4.6.1.2 Response

bin_id:

Identifies the Bin.

content:

The Bin's content.

target_grid:

The bin's target Grid.

4.6.1.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getbininfo" },
   element params {
      ( element bin_id { xsd:int { minInclusive = "1" } } ,
       | ( element min_bin_id { xsd:int { minInclusive = "1" } } ?,
           element max_bin_id { xsd:int { minInclusive = "1" } } ? )
      ) ?
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element bins {
         element bin {
            element bin_id { xsd:int { minInclusive = "1" } },
            element content { xsd:short { minInclusive = "1" } },
            element target_grid { xsd:short { minInclusive = "1" } }
         } +
      } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getbininfo</name>
   <params>
      <bin_id>12004</bin_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin>
            <bin_id>12004</bin_id>
            <content>21</content>
            <target_grid>1</target_grid>
         </bin>
      </bins>
   </params>
</response>
```

4.6.1.4 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1049	Response overflow.

4.6.2 Get Bin Location

The **getbinloc** method retrieves information about the physical location of the Bins in the system.

4.6.2.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only information about that specific Bin is returned.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only bins with bin_id greater than or equal to min_bin_id are returned.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only bins with bin_id less than or equal to max_bin_id are returned.

4.6.2.2 Response

bin_id:

Identifies the Bin.

grid_id:

The actual Grid where the Bin is located.

bins_above:

The number of physical Bins that are stacked above the Bin.

xpos:

The physical X position of the Bin.

ypos:

The physical Y position of the Bin.

depth:

The physical depth in mm of the Bin.

bin_mode:

The current Bin mode.

4.6.2.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getbinloc" },
   element params {
      ( element bin_id { xsd:int { minInclusive = "1" } } ,
       | ( element min_bin_id { xsd:int { minInclusive = "1" } } ?,
           element max_bin_id { xsd:int { minInclusive = "1" } } ?
         )
      ) ?
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element bins {
         element bin {
            element bin_id { xsd:int { minInclusive = "1" } },
            element grid_id { xsd:short { minInclusive = "1" } },
            element bins_above { xsd:int } ?,
            element xpos { xsd:short { minInclusive = "1" } } ?,
            element ypos { xsd:short { minInclusive = "1" } } ?,
            element depth { xsd:short } ?,
            element bin_mode { "GRID" | "PREPARED" | "PORT" |
"TOGRID" | "OPEN" | "CLOSED" | "OUTSIDE" }
         } +
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getbinloc</name>
   <params>
      <bin_id>12004</bin_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin>
            <bin_id>12004</bin_id>
            <grid_id>1</grid_id>
            <bins_above>1</bins_above>
            <xpos>21</xpos>
            <ypos>21</ypos>
            <depth>362</depth>
            <bin_mode>GRID</bin_mode>
         </bin>
      </bins>
   </params>
</response>
```

4.6.2.4 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1049	Response overflow.

4.6.3 Get Bin State

The **getbinstate** method retrieves information about the Bins in the system. The method will be extended with additional response elements inside <bin>.

4.6.3.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only information about that specific Bin is returned.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only information about Bins with bin_id greater than or equal to min_bin_id is returned.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only information about Bins with bin_id less than or equal to max_bin_id is returned.

4.6.3.2 Response

bin_id:

Identifies the Bin.

content:

The content of the Bin.

target_grid:

The preferred storage Grid for the Bin.

xpos:

The physical X position of the Bin.

ypos:

The physical Y position of the Bin.

depth:

The physical depth in mm of the Bin.

port_id:

The port in which the Bin is located. (Numeric 0 if the Bin is not in a Port.)

bin_mode:

The current Bin mode, same codes as from LogPublisher.

4.6.3.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getbinstate" },
   element params {
      ( element bin_id { xsd:int { minInclusive = "1" } } ?,
         | ( element min_bin_id { xsd:int { minInclusive = "1" } } ?,
         element max_bin_id { xsd:int { minInclusive = "1" } } ?
         )
```

```
      ) ?
   }
}
```

Compact syntax (Response):

```
element response {
   element bins {
      element bin {
         element bin_id { xsd:int { minInclusive = "1" } },
         element content { xsd:short { minInclusive = "1" } },
         element target_grid { xsd:short { minInclusive = "1" } },
         element xpos { xsd:short { minInclusive = "1" } },
         element ypos { xsd:short { minInclusive = "1" } },
         element depth { xsd:short },
         element port_id { xsd:int { minInclusive = "1" } },
         element bin_mode { "G" | "T" | "P" | "R" | "O" | "C" | "X" |
            "F" }
      } +
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getbinstate</name>
   <params>
      <bin_id>10000</bin_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin>
            <bin_id>10000</bin_id>
            <content>1</content>
            <target_grid>1</target_grid>
            <xpos>34</xpos>
            <ypos>38</ypos>
            <depth>1124</depth>
            <port_id>0</port_id>
            <bin_mode>G</bin_mode>
         </bin>
      </bins>
   </params>
</response>
```

4.6.3.4 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1049	Response overflow.

1055	Parameter min_bin_id is not valid (invalid format or out of range).
1056	Parameter max_bin_id is not valid (invalid format or out of range).

4.6.4 Get Task Information

The **gettaskinfo** message requests information about the tasks in the system.

4.6.4.1 Parameters

taskgroup_id: (optional)

If the parameter taskgroup_id is provided, only information about that task group is returned.

min_taskgroup_id: (optional)

If the parameter min_taskgroup_id is provided, only task groups with taskgroup_id greater than or equal to min_taskgroup_id are returned.

max_taskgroup_id: (optional)

If the parameter max_taskgroup_id is provided, only task groups with taskgroup_id less than or equal to max_taskgroup_id are returned.

task_id: (optional)

If the parameter task_id is provided, only information about that task is returned. If a task_id is specified, then a taskgroup_id should not be specified.

4.6.4.2 Response

taskgroup_id:

Identifies the task group.

shipment:

The shipment the task group belongs to.

category:

The category the task group belongs to.

req_time:

The task group's req_time property.

priority:

The task group's priority property.

start_time:

The task group's start_time property.

suspended:

Tells if the task group has been suspended or not.

tasks:

One or more tasks that belong to the specific task group.

Each task element contains the following information:

task_id:

Identifies the specific task.

bin_id:

The Bin which belongs to this task.

item:

The task's item property.

order:

The task's order property.

estpicktime:

The task's estimated handling time, specified in seconds.

4.6.4.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "gettaskinfo" },
   element params {
      ( element task_id { xsd:int { minInclusive = "1" } }
      | element taskgroup_id { xsd:int { minInclusive = "1" } }
      | ( element min_taskgroup_id { xsd:int { minInclusive = "1"
         } } ?,
         element max_taskgroup_id { xsd:int { minInclusive = "1"
            } } ? )
      ) ?
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element taskgroup {
         element taskgroup_id { xsd:int { minInclusive = "1" } },
         element shipment { xsd:int { minInclusive = "1" } } ?,
         element category { xsd:short { minInclusive = "1" } },
         element req_time { xsd:dateTime } ,
         element priority { xsd:short { minInclusive = "1" } },
         element start_time { xsd:dateTime } ?,
         element suspended { xsd:boolean } ?,
         element tasks {
            element task {
               element task_id { xsd:int { minInclusive = "1" } },
               element bin_id { xsd:int { minInclusive = "1" } },
               element item { xsd:int { minInclusive = "1" } } ?,
               element order { xsd:int { minInclusive = "1" } } ?,
               element estpicktime { xsd:int { minInclusive =
                  "1" } } ?
            } +
         }
      } +
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>gettaskinfo</name>
```

```
   <params>
      <taskgroup_id>134523</taskgroup_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <taskgroup>
         <taskgroup_id>134523</taskgroup_id>
         <shipment>20961</shipment>
         <category>1</category>
         <req_time>2009.03.20 14:00</req_time>
         <priority>20</priority>
         <tasks>
            <task>
               <task_id>420192</task_id>
               <bin_id>10304</bin_id>
            </task>
            <task>
               <task_id>420193</task_id>
               <bin_id>33570</bin_id>
            </task>
         </tasks>
      </taskgroup>
   </params>
</methodcall>
```

4.6.4.4 Error Codes

Error Code	Description
1012	Parameter task_id is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1017	The specified task does not exist.
1018	The specified task group does not exist.
1023	Multiple selection parameters are not allowed.
1049	Response overflow.

4.6.5 Get Next Task Groups

The **getnexttaskgroups** message requests information about which task groups that is planned for a port.

The same information is published in the Log Publisher PTG segment.

With AS Router, the WMS must activate this information for each port by using the **configureport** method. By default, no information is published. The WMS can set the required lookahead by setting the value *nextTaskGroupBinLookahead*. The next task groups that will be started within that number of bins/tasks will be published. Please note that AS Router should not change/optimize published information, so enabling this feature may have a performance impact (depending on lookahead).

With AS Planner, the WMS cannot specify the lookahead length. On a carousel port, the look-ahead visibility depends on situation but might be 1-4 bins ahead. On a relay port, the visibility is normally longer. We do not publish the next bins, only the next task groups, so there's no visibility until you approach the end of the current task group.

In rare cases other task groups can be delivered (for instance if the Planner/Router/Driver is manually restarted).

4.6.5.1 Parameters

port_id: (mandatory)

Specifies the port.

4.6.5.2 Response

taskgroups

Ordered list of the next task groups that is planned delivered in the specified port.

taskgroup_id:

Identifies a task group.

4.6.5.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getnexttaskgroups" },
   element params {
      element port_id { xsd:int { minInclusive = "1" } }
   }
}
```

Compact syntax (Response):

```
element response {
   element <params>
      element taskgroups {
         element taskgroup_id { xsd:int { minInclusive = "1" } } *
      }
    }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getnexttaskgroups</name>
   <params>
      <port_id>1</port_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <taskgroups>
```

```
            <taskgroup_id>225143</taskgroup_id>
            <taskgroup_id>211020</taskgroup_id>
        </taskgroups>
    </params>
</response>
```

4.6.5.4 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).

4.7 Controlling the Sequence of Task Groups

The WMS can set priority on the task groups, using the properties *req_time* and *priority*. AutoStore will normally follow this prioritization when preparing and delivering bins. However, AutoStore does not follow this priority strictly. AutoStore may deliver lower prioritized tasks in order to achieve a higher throughput. It is normally better to keep the ports busy working (and deliver a low priority task group) instead of waiting for the highest prioritized task group which may be unavailable at the moment (e.g. if the bin is in use in another port, or not yet prepared).

Using the standard prioritization concept is normally a very good way to balance between getting high performance and at the same time delivering the most important task groups first. This section describes additional features and concepts that can be used to further control the sequence of task groups.

4.7.1 Overriding the next Task Group

The WMS may explicitly select a task group with the *openbin* method. This is not recommended during normal operation. AutoStore plans a flow of bins to a port, and overriding the plan in the last minute will cause a performance penalty. The planned bins may already be moved to the port and may need to be moved away.

The use case of explicitly selecting a task group with openbin is when the WMS wants to continue on a suspended task group. In this case, the openbin will immediately follow the openport. When a user logs out (closes the port) in the middle of a task group, the WMS normally specifies that the current task group shall be suspended. (If the task group is not suspended, the task group could be selected and continued on another port.) To continue on the task group on the same port when the user logs in, the WMS can explicitly select the suspended task group.

4.7.2 Implementing Cut-Off Time

A warehouse may have different cut-off times during a day, and the different task groups may be targeting specific deadlines. A normal prioritization concept would be to set the task groups' *req_time* equal to their deadline. This would give the task groups with the earliest deadline highests priority, and would in many cases be the preferred solution. But in order to optimize the total throughput, it is still allowed to pick task groups with later deadlines.

To fully prioritize picking of the upcoming deadline, it is possible to specify that a port only selects task groups with *req_time* equal to or lower than a specified cut-off time.

The port's req_time limit (see Select Requirements on page 24) is set by using the req_time parameter in the *openport* method.

4.7.3 Synchronizing with the Task Group Sequence

Sometimes, the WMS needs to synchronize external processes, for instance package machines, with the task groups that are being picked in the port. In these use cases, the WMS does not need to decide the exact sequence of task groups at the ports, but the WMS needs visibility into the exact sequence of task group sequence that AutoStore will deliver.

The method *getnexttaskgroups* (see Get Next Task Groups on page 53) returns information about which task groups will be delivered to a port. The same information is available from Log Publisher. Different processes and installations may need different length of visibility into the future. Sometimes the WMS only needs to know the next task group half a minute in advance, while in other cases the WMS may need to know the upcoming task groups many minutes in advance. Therefore, it is possible to configure this length of visibility per port.

Please note that AS Router should not change/optimize the published information, so configuring a long look-ahead horizon may have a performance impact (depending on the length). However, this should normally not be a problem and compared to explicitly overriding the sequence from the WMS side it is probably better to let AutoStore decide the sequence.

4.7.4 Specifying a Strict Sequence of Task Groups

Some use cases may require the WMS to specify the sequence of the task groups at ports, for example in order to consolidate bins from different ports or grids within a limited time interval.

With the introduction of port mode SEQUENCE, it is possible to specify the exact sequence of task group to be delivered at the ports.

> **NOTICE**
> **The port mode SEQUENCE requires the April 2023 software release**

4.7.4.1 Port Mode SEQUENCE

When a port is opened in the special port mode *SEQUENCE*, the WMS can specify the exact sequence of task groups to be delivered. (This is similar to the Port Queue concept of the Bin Interface, but the sequence feature in Task Interface operates on task groups instead of bins.) In order to benefit from the AutoStore optimizations, this feature should only be used when it is required, and only for the ports that require a specific sequence of task groups.

At the same time, some ports can be opened in mode SEQUENCE while other ports can operate in the standard OPEN mode.

The WMS specifies the sequence of task groups to be delivered by using the methods *setportsequence* and *appendportsequence*.

To avoid conflicts between the sequences specified by the WMS and the normal delivery of task groups to ports in the other modes (e.g. OPEN/SWAP/AUTO), the WMS must use different set if categories for the task groups used in mode SEQUENCE than for the task groups used in the other port modes. There are also other limitations for the task groups that are used in a sequence.

The following rules apply:

- *openport* (mode OPEN/AUTO/SWAP) with a list of categories is rejected if another Port's sequence already includes a task group with same category as one of the listed categories.
- *setportsequence/appendportsequence* is rejected if another port is opened (mode OPEN/AUTO/SWAP) with same category as any of the specified task groups.
- Task groups that shall be used in a port's sequence cannot have a *shipment* property.
- A task group cannot be added to different ports' sequence at the same time.
- Updating a task group (*updatetaskgroup*) that belongs to a port's sequence is not allowed.
- Adding tasks (*addtask*) to a task group that belongs to a port's sequence is allowed (standard rules apply).
- Cancelling a task group (*cancel_taskgroup*) will remove it from the port's sequence.
- *closeport* clears the port's sequence.

- *openport* with mode SEQUENCE shall not specify any categories or content parameters.

The SEQUENCE feature should be used with caution. Consider if the standard port mode can be used (e.g. looking into the port sequence instead of specifying it, or using cutoff restriction on the port).

Consider the following guidelines when using the SEQUENCE feature:

- Unprepared task groups will cause waiting in port and wasted performance. Use Log Publisher (alterntively *getbinlocation*) to check the preparation state of the bins before adding the task group to a port's sequence queue.
- Use long enough sequence queues to allow AutoStore to plan for the bin delivery. (10-15 minutes of operation.)
- Avoid bin conflicts between ports. Do not plan same bin to arrive on different ports at the same time.
- If alternative ports can be used for a task group, consider the travelling distance between port and bins.

4.7.4.2 Set Port Sequence

The **setportsequence** method specifies the sequence of task group for a given port.

The current port sequence is cleared, and replaced by the specified sequence of task groups. Specifying an empty sequence clears the port sequence.

In the normal work flow, the WMS should use *appendportsequence* to add task groups to the port sequence. The *setportsequence* method should be used with caution, because changes to the ongoing plans will have a performance penalty.

> **NOTICE**
> **Updating the Port Sequence does not affect the selected_taskgroup. The ongoing task group will still be delivered to the port.**

4.7.4.2.1 Parameters

port_id: (mandatory)

Identifies the Port.

taskgroups: (mandatory)

Encloses the task group sequence.

taskgroup_id: (optional)

Identifies the task group.

4.7.4.2.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "setportsequence" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element taskgroups {
         ( element taskgroup_id { xsd:int { minInclusive = "1" } }
         ) +
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>setportsequence</name>
   <params>
      <port_id>4</port_id>
      <taskgroups>
         <taskgroup_id>137802</taskgroup_id>
         <taskgroup_id>144911</taskgroup_id>
         <taskgroup_id>202554</taskgroup_id>
         <taskgroup_id>162597</taskgroup_id>
      </taskgroups>
   </params>
</methodcall>
```

4.7.4.2.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1018	The specified task group does not exist.
1044	The task group is selected by a Port.
1090	Category conflict.
1091	The task group belongs to a shipment.
1092	The task group belongs to a sequence.

4.7.4.3 Append Port Sequence

The **appendportsequence** method appends task groups to the port's sequence queue.

4.7.4.3.1 Parameters

port_id: (mandatory)

Identifies the Port.

taskgroups: (mandatory)

Encloses the task group sequence.

taskgroup_id: (mandatory)

Identifies the task group.

4.7.4.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "appendportsequence" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element taskgroups {
         ( element taskgroup_id { xsd:int { minInclusive = "1" } }
         ) +
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>appendportsequence</name>
   <params>
      <port_id>4</port_id>
      <taskgroups>
         <taskgroup_id>133204</taskgroup_id>
         <taskgroup_id>131445</taskgroup_id>
      </taskgroups>
   </params>
</methodcall>
```

4.7.4.3.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1018	The specified task group does not exist.
1031	Missing parameter taskgroup_id.
1044	The task group is selected by a Port.
1090	Category conflict.
1091	The task group belongs to a shipment.
1092	The task group belongs to a sequence.

5. Bin Interface

5.1 Basic Concepts

5.1.1 Bin Preparation

A **prepared** Bin is available for the robots without additional digging or preparation.

A prepared Bin is protected against being dug down or covered by other work areas. The Bin will remain available for the Robots as long as it stays in the prepared state.

A prepared Bin is normally placed in position as the upper Bin in its stack. Most of the upper Bins in a Grid are not prepared, and may become blocked by other digging areas. The number of simultaneous prepared Bins is limited, because each prepared Bin will occupy an area on the Grid. Therefore, in rare cases, a prepared Bin may temporarily leave the prepared state (this will not affect its place in the preparation queue).

The WMS requests AutoStore to prepare Bins. The requested Bins constitute a "preparation queue".

The Bins in the preparation queue are in a prioritized order. However, AutoStore will try to prepare every Bin within a reasonable time.

It is recommended that the preparation queue contains Bins for the next 30 minutes of operation.

A Bin may appear multiple times in the preparation queue. The WMS may request the Bin being prepared once for each planned picking operation.

The foremost appearance of a Bin leaves the preparation queue when the Bin is closed or removed from a Port.

A selected Bin (returned by the *openbin* method) has the highest priority. If the WMS explicitly selects a Bin as an *ad hoc* operation (for instance in a short pick or deviation situation), it should normally first request a preparation (even if it won't wait for it to be prepared) in order to avoid releasing the Bin preparation request belonging to another picking operation.

The rearmost appearance of a Bin leaves the preparation queue when a preparation request is canceled.

5.1.2 Port Queue

Each Port has its own Port queue.

The Port queue is maintained by the WMS, and specifies the sequence of Bins that AutoStore should plan for each Port. The Port queue also specifies which Bins AutoStore selects on *openbin* requests.

Normally, the WMS will put prepared Bins in the Port queue, but this is not a requirement. The Port queue is typically not very long. The WMS will normally defer this decision in order to avoid conflicts between Ports (same bin required) and to consider preparation status.

The Bins in the Port queue are prioritized by their order (the sequence in which they are needed). Normally the priority of the Bins in the Port queue is higher than the preparation queue, except for the backmost Bins of a very long Port queue.

It is recommended to keep a Port queue that covers the next approximate 10-15 minutes of operation.

The Port queue may refer to content codes.

When an element in the Port queue refers to a content code, AutoStore will select a Bin with the specified content code. AutoStore will never select a Bin that already exists in any Port queue or in the preparation queue to replace a content code. When AutoStore selects a Bin on the *openbin* request where a content code is the front element in the Port queue, then the selected bin_id replaces the content code as the front element in the Port queue.

The front Bin leaves the Port queue when the Bin is closed in the Port. In Port mode **AUTO**, the front Bin leaves the Port queue when the Bin is confirmed to Port.

Only the front bin is removed from the port queue. If another bin in the middle of the port queue is explicitly selected by the WMS, it will not be removed from the port queue when it is closed.

Although a Bin normally leaves both the preparation queue and the Port queue simultaneously, these queues are independent of each other.

5.1.2.1 Port Queue Group

It is possible for the WMS to specify that bins in the port queue belong together in groups. Bins within one such group can have their sequence changed by AutoStore to optimize performance. It is still important that the WMS plans and optimizes the Port Queue sequence, because AutoStore will plan the delivery according to the given sequence. But when an *openbin* command is received, AutoStore will evaluate if another bin can be delivered faster than the first one. The *openbin* response includes the selected bin.

Port queue groups cannot be nested inside each other.

The *begingroupportqueue* method will implicitly end the current group, and start a new group. The bins appended to the port queue after *begingroupportqueue* will belong to the same group, until the group is ended with *endgroupportqueue* or *begingroupportqueue*. Bins are not required to belong to a group. Bins appended after *endgroupportqueue* do not belong to a group, and will be delivered strictly in the specified sequence.

Example response from *getportqueue* (with comments):

```
<?xml version="1.0" encoding="utf-8"?>
<response> <params> <bins>
   <bin_id>11</bin_id> -- outside group, will be delivered first
   <begingroup />
   <bin_id>101</bin_id> -- 101, 102, 103 are in the same group
   <bin_id>102</bin_id>
   <bin_id>103</bin_id>
   <begingroup />
   <bin_id>201</bin_id> -- 201, 202, 203 are in the same group
   <bin_id>202</bin_id>
   <bin_id>203</bin_id>
   <endgroup />
   <bin_id>12</bin_id> -- outside group (specified sequence)
   <bin_id>13</bin_id> -- outside group (specified sequence)
   <begingroup />
   <bin_id>401</bin_id> -- 401, 402, 403 are in the same group
   <bin_id>402</bin_id> -- group is not yet terminated,
   <bin_id>403</bin_id> -- appendportqueue into same group
</bins> </params> </response>
```

Please note that AutoStore removes superfluous begingroup/endgroup elements.

5.2 Port Methods

5.2.1 Open Port

The **openport** method declares that a port is ready for work.

5.2.1.1 Parameters

port_id: (mandatory)

Identifies the port.

port_mode: (optional)

Specifies a special mode of operation.

The default port mode is **OPEN**, unless a special mode is specified.

5.2.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "openport" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element port_mode { "INSERT" | "SWAP" | "MOVE" | "AUTO" } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>openport</name>
   <params>
      <port_id>4</port_id>
   </params>
</methodcall>
```

5.2.1.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1002	The specified Port is already open.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1048	System is not available.
1060	Parameter port_mode is not valid (invalid format or out of range).

5.2.2 Open Bin

The *openbin* method requests a bin in the port.

AutoStore selects a bin according to the the port's queue.

When AutoStore selects a bin according to a content code, the front element in the port's queue is changed from content code to the selected bin.

The WMS may explicitly select a specific bin.

The WMS cannot explicitly select a bin that conflicts with the current selected bin (a previous openbin call that is not yet closed).

For optimal performance, AutoStore should select the bin (specified by the port queue). Explicit selects will inflict a performance penalty, especially if the bin is not yet prepared.

Do not include the expected bin_id as parameter, you should rather check that the response is the expected bin_id. If the WMS and AutoStore is out of sync of some reason, including a bin_id parameter will cause huge delays because it will always override what AutoStore selects.

5.2.2.1 Parameters

port_id: (mandatory)

Identifies the port.

bin_id: (optional)

Explicitly selects a specific bin.

5.2.2.2 Response

port_id:

Identifies the port.

bin_id:

The port's selected bin.

5.2.2.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "openbin" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } } ?
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>openbin</name>
   <params>
      <port_id>4</port_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
```

```
    <params>
       <port_id>4</port_id>
       <bin_id>14861</bin_id>
    </params>
</response>
```

5.2.2.4 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1015	There are no tasks or Bins that fulfill the Port select requirements.
1016	There are no Bins ready that fulfill the Port select requirements.
1020	The openbin conflicts with a previous openbin call that is not yet closed.
1048	System is not available.

5.2.3 Close Bin

The **closebin** method closes the bin in the port.

The bin must be opened and ready in the port before the bin can be closed. The WMS must specify the correct bin when it closes a bin.

The WMS may update the bin's content and target Grid.

If the selected bin is the front bin in the port queue, then the *closebin* method removes the front bin from the port queue.

If the selected bin is in the preparation queue, then the *closebin* method removes the foremost appearance of the selected bin from the preparation queue.

5.2.3.1 Parameters

port_id: (mandatory)

Identifies the port.

bin_id: (mandatory)

Must be equal to the open bin.

content: (optional)

Sets the bin's content.

If this parameter is not provided, the bin's content is not changed.

target_grid: (optional)

Specifies the target Grid of the bin.

If this parameter is not provided, the bin's target Grid is not changed.

5.2.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "closebin" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } },
      element content { xsd:short { minInclusive = "1" } } ?,
      element target_grid { xsd:short { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>closebin</name>
   <params>
      <port_id>4</port_id>
      <bin_id>14861</bin_id>
      <target_grid>2</target_grid>
   </params>
</methodcall>
```

5.2.3.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1024	Mandatory parameter bin_id is missing.
1026	The Port does not have a selected Bin.
1027	The Bin is not yet ready in the Port.
1028	The Bin and task parameters conflict with the open Bin and task.
1048	System is not available.
1061	Parameter target_grid is not valid (invalid format or out of range).
1083	No room for given content.

5.2.4 Close Port

The **closeport** method closes the port.

If there is an open bin in the port, the bin is closed. The bin's content and target Grid is not updated.

The port's queue and the preparation queue are not changed by the *closeport* method.

The WMS normally also clears the port's queue when a port is closed. However, if the port is closed temporarily because the warehouse operator takes a short break, keeping the port queue will ensure that the bins are still being planned for the port.

In the case of a multi-Grid installation, if the port is not going to be used for a while, the WMS normally reopens the port in **SWAP** mode so that AutoStore may use the port for the background process that exchanges bins between the Grids.

The port's selected bin is cleared by the *closeport* method.

5.2.4.1 Parameters

port_id: (mandatory)

Identifies the Port.

5.2.4.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "closeport" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<name>closeport</name>
   <params>
      <port_id>4</port_id>
   </params>
</methodcall>
```

5.2.4.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1048	System is not available.

5.2.5 Insert Bin

The **insertbin** method declares that a bin has been inserted in a port and closes the inserted bin.

The port cannot have a selected bin.

The WMS may update the bin's content.

5.2.5.1 Parameters

port_id: (mandatory)

Identifies the port.

bin_id: (mandatory)

Identifies the bin.

target_grid: (optional)

Sets the target storage Grid for the bin. If this parameter is omitted, the bin will keep its previous target Grid.

content: (optional, but mandatory when bin_type is set)

Sets the bin's content.

If this parameter is omitted, the bin will keep its previous content. If the bin has never been in the system before, the content is set to "0".

bin_type: (optional, but mandatory if the bin is not defined yet)

Specifies the bin's type.

This parameter is mandatory if the bin is not defined (the first time the bin is inserted). In installations where bins are removed and inserted, the bin_type parameter is not required in subsequent inserts.

5.2.5.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "insertbin" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } },
      element content { xsd:short { minInclusive = "1" } } ?,
      element bin_type { xsd:short { minInclusive = "1" } } ?,
      element target_grid { xsd:short { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>insertbin</name>
   <params>
      <port_id>4</port_id>
      <bin_id>16477</bin_id>
      <content>10</content>
   </params>
</methodcall>
```

5.2.5.3 Error Code

Error Code	Description

1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1029	The specified Bin is already inserted.
1030	The Port has a selected Bin.
1048	System is not available.
1058	Bin is still outside.
1061	Parameter target_grid is not valid (invalid format or out of range).
1073	Parameter bin_type is not valid (invalid format or out of range).
1077	Maximum number of Bins already inserted.
1083	No room for given content.

5.2.6 Remove Bin

The **removebin** method declares that the open bin has been removed from a port. The specified bin must be open and ready in the port.

A removed bin will be assigned bin mode **OUTSIDE**. The bin will keep its current content, bin_type and target_grid.

The *removebin* method updates the port queue and the preparation queue in the same way as the *closebin* method.

If the selected bin is the front bin in the port queue, then the *removebin* method removes the front bin from the port queue.

If the selected bin is in the preparation queue, then the *removebin* method removes the foremost appearance of the selected bin from the preparation queue.

5.2.6.1 Parameters

port_id: (mandatory)

Identifies the port.

bin_id: (mandatory)

Must be equal to the open bin in the port.

5.2.6.2 Message Format

Compact syntax:

```
element methodcall {
```

```
      element name { "removebin" },
      element params {
         element port_id { xsd:short { minInclusive = "1" } },
         element bin_id { xsd:int { minInclusive = "1" } }
      }
   }
```

Request example:

```
<?xml version="1.0"?>
<name>removebin</name>
<params>
   <port_id>4</port_id>
   <bin_id>14861</bin_id>
</params>
</methodcall>
```

5.2.6.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1024	Mandatory parameter bin_id is missing.
1025	Parameter taskcomplete is not valid (invalid format or out of range).
1026	The Port does not have a selected Bin.
1027	The Bin is not yet ready in the Port.
1028	The Bin and task parameters conflict with the open Bin and task.
1048	System is not available.

5.2.7 Update Bin

The **updatebin** method updates the content of a bin.

5.2.7.1 Parameters

bin_id: (mandatory)

Identifies the Bin that should be updated.

content: (optional)

Sets the Bin's content.

target_grid: (optional)

Sets the Bin's target Grid.

5.2.7.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "updatebin" },
   element params {
      element bin_id { xsd:int { minInclusive = "1" } },
      element content { xsd:short { minInclusive = "1" } } ?,
      element target_grid { xsd:short { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>updatebin</name>
   <params>
      <bin_id>14861</bin_id>
      <content>11</content>
   </params>
</methodcall>
```

5.2.7.3 Error Code

Error Code	Description
1006	Parameter content is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1024	Mandatory parameter bin_id is missing.
1061	Parameter target_grid is not valid (invalid format or out of range).

5.3 Bin Preparation Methods

5.3.1 Prepare Bin

The **preparebin** method requests one or more bins to be prepared for use.

The method appends the bins to the existing preparation queue.

5.3.1.1 Parameters

bins: (mandatory)

Encloses the bin sequence.

bin_id: (mandatory)

Identifies a bin.

5.3.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "preparebin" },
   element params {
      element bins {
         element bin_id { xsd:int { minInclusive = "1" } }+
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>preparebin</name>
   <params>
      <bins>
         <bin_id>21334</bin_id>
         <bin_id>25860</bin_id>
      <bins>
   </params>
</methodcall>
```

5.3.1.3 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1048	System is not available.
1063	Bin preparation queue is full.
	The maximum size of the bin preparation queue is 20000 bins.
1064	Mandatory parameter element Bins are missing.

5.3.2 Insert Prepare Bin

The **insertpreparebin** method inserts one or more bins into the preparation queue at a specified index.

The first position in the queue is Index = 1.

5.3.2.1 Parameters

index: (mandatory)

Specifies the index where the bins are inserted.

bins: (mandatory)

Encloses the bin sequence.

bin_id: (mandatory)

Identifies a bin.

5.3.2.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "insertpreparebin" },
   element params {
      element index { xsd:int { minInclusive = "1" } }
      element bins {
         element bin_id { xsd:int { minInclusive = "1" } }+
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>insertpreparebin</name>
   <params>
      <index>35</index>
      <bins>
         <bin_id>21334</bin_id>
         <bin_id>25860</bin_id>
      <bins>
   </params>
</methodcall>
```

5.3.2.3 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1048	System is not available.
1063	Bin preparation queue is full.
	The maximum size of the bin preparation queue is 20000 bins.
1064	Mandatory parameter element Bins are missing.
1081	Mandatory parameter index is missing.
1082	Parameter index not valid (invalid format or out of range).

5.3.3 Cancel Prepare Bin

The **cancelpreparebin** method cancels previous preparation requests.

The specified bins are canceled. The rearmost appearance of a bin leaves the preparation queue when a preparation request is canceled.

It is possible to specify that all bins should be canceled without listing the bins.

Canceling preparation requests does not affect the port queues. These queues are independent of each other.

5.3.3.1 Parameters

all: (optional)

Specifies that all bins should be removed from the preparation queue.

bins:

Encloses the bin sequence.

bin_id:

Identifies the bin.

5.3.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "cancelpreparebin" },
   element params {
      element all { xsd:boolean },
      | element bins {
         element bin_id { xsd:int { minInclusive = "1" } }+
        }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>cancelpreparebin</name>
   <params>
      <bins>
         <bin_id>21334</bin_id>
      </bins>
   </params>
</methodcall>
```

5.3.3.3 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1048	System is not available.
1065	Parameter all is not valid (invalid format or out of range).

5.3.4 Set Port Queue

The **setportqueue** method specifies the Port queue.

The current Port queue is cleared, and replaced by the specified Bin sequence.

The WMS may clear the Port queue by specifying an empty Bin sequence. There is no *cancelportqueue* method to remove Bins from the Port queue. Sine the Port queue is normally short, the WMS should just replace the entire queue.

Clearing a Port queue does not affect the preparation queue. These queues are independent of each other.

In the normal work flow, the WMS should use *appendportqueue* to add Bins to the Port queue. The *setportqueue* method should be used with caution, because changes to the Port queues (ongoing plans) will have a performance penalty.

> **NOTICE**
> **Updating the Port queue does not affect the selected_bin. The selected_bin will be delived to port and is not affected by changes in the port queue. The only way to cancel a selected_bin before it is delivered is to close the port.**

5.3.4.1 Parameters

port_id: (mandatory)

Identifies the Port.

bins: (mandatory)

Encloses the Bin sequence.

bin_id: (optional)

Identifies the Bin.

content: (optional)

Specifies a content code.

The sequence of the bin_id and content elements specify the Bin sequence.

begingroup: (optional)

Starts a group in the Port queue.

endgroup: (optional)

Ends a group in the Port queue.

5.3.4.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "setportqueue" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bins {
         ( element bin_id { xsd:int { minInclusive = "1" } }
         | element content { xsd:short { minInclusive = "0" } }
         | element begingroup { empty }
         | element endgroup { empty }
         ) +
      }
   }
```

```
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>setportqueue</name>
   <params>
      <port_id>4</port_id>
      <bins>
         <bin_id>35432</bin_id>
         <begingroup/>
         <bin_id>20443</bin_id>
         <bin_id>34813</bin_id>
         <endgroup/>
         <bin_id>40602</bin_id>
         <bin_id>45977</bin_id>
      <bins>
   </params>
</methodcall>
```

5.3.4.3 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1048	System is not available.
1062	Port queue is full.
	The maximum size of a port's port queue is 1000 bins.

5.3.5 Append to Port Queue

The **appendportqueue** method appends Bins to the Port queue.

5.3.5.1 Parameters

port_id: (mandatory)

Identifies the Port.

bins: (mandatory)

Encloses the Bin sequence. The sequence of the bin_id and content elements specify the Bin sequence.

bin_id: (optional)

Identifies the Bin.

content: (optional)

Specifies a content code.

begingroup: (optional)

Starts a group in the Port queue.

endgroup: (optional)

Ends a group in the Port queue.

5.3.5.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "appendportqueue" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bins {
         ( element bin_id { xsd:int { minInclusive = "1" } }
         | element content { xsd:short { minInclusive = "0" } }
         | element begingroup { empty }
         | element endgroup { empty }
         ) +
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>appendportqueue</name>
   <params>
      <port_id>4</port_id>
      <bins>
         <bin_id>10343</bin_id>
      <bins>
   </params>
</methodcall>
```

5.3.5.3 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1048	System is not available.
1062	Port queue is full.
	The maximum size of a port's port queue is 1000 bins.

5.3.6 Begin Group in Port Queue

The **begingroupportqueue** method starts a new group in the port queue. It implicitly ends the current group.

5.3.6.1 Parameters

port_id: (mandatory)

Identifies the port.

5.3.6.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "begingroupportqueue" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>begingroupportqueue</name>
   <params>
      <port_id>4</port_id>
   </params>
</methodcall>
```

5.3.6.3 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1048	System is not available.
1062	Port queue is full.
	The maximum size of a port's port queue is 1000 bins.

5.3.7 End Group in Port Queue

The **endgroupportqueue** method ends the current group in the port queue.

5.3.7.1 Parameters

port_id: (mandatory)

Identifies the port.

5.3.7.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "endgroupportqueue" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>endgroupportqueue</name>
   <params>
      <port_id>4</port_id>
   </params>
</methodcall>
```

5.3.7.3 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1048	System is not available.
1062	Port queue is full.
	The maximum size of a port's port queue is 1000 bins.

5.4 Query Methods

5.4.1 Get Bin Information

The **getbininfo** method retrieves information about the bins in the system.

5.4.1.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only information about that specific bin is returned.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only bins with bin_id greater than or equal to min_bin_id are returned.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only bins with bin_id less than or equal to max_bin_id are returned.

5.4.1.2 Response

bin_id:

Identifies the bin.

content:

The bin's content.

target_grid:

The bin's target Grid.

5.4.1.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getbininfo" },
   element params {
      element bin_id { xsd:int { minInclusive = "1" } } ?,
      element min_bin_id { xsd:int { minInclusive = "1" } } ?,
      element max_bin_id { xsd:int { minInclusive = "1" } } ?
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element bins {
         element bin {
            element bin_id { xsd:int { minInclusive = "1" } },
            element content { xsd:short { minInclusive = "1" } },
            element target_grid { xsd:short { minInclusive = "1" } }
         } +
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getbininfo</name>
   <params>
      <bin_id>12004</bin_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin>
            <bin_id>12004</bin_id>
            <content>21</content>
            <target_grid>1</target_grid>
         </bin>
      </bins>
   </params>
```

```
</response>
```

5.4.1.4 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1049	Response overflow.
1055	Parameter min_bin_id is not valid (invalid format or out of range).
1056	Parameter max_bin_id is not valid (invalid format or out of range).

5.4.2 Get Bin Location

The **getbinloc** method retrieves information about the physical location of the Bins in the system.

5.4.2.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only information about that specific Bin is returned.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only Bins with bin_id greater than or equal to min_bin_id are returned.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only Bins with bin_id less than or equal to max_bin_id are returned.

5.4.2.2 Response

bin_id:

Identifies the Bin.

grid_id:

The actual Grid where the Bin is located.

xpos:

The physical X position of the Bin.

ypos:

The physical Y position of the Bin.

depth:

The physical depth in mm of the Bin.

bin_mode:

The current Bin mode.

5.4.2.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getbinloc" },
   element params {
      element bin_id { xsd:int { minInclusive = "1" } } ?,
      element min_bin_id { xsd:int { minInclusive = "1" } } ?,
      element max_bin_id { xsd:int { minInclusive = "1" } } ?,
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element bin {
         element bin_id { xsd:int { minInclusive = "1" } },
         element grid_id { xsd:short { minInclusive = "1" } },
         element xpos { xsd:short { minInclusive = "1" } } ?,
         element ypos { xsd:short { minInclusive = "1" } } ?,
         element depth { xsd:short } ?,
         element bin_mode { "GRID" | "PREPARED" | "PORT" | "TOGRID" |
"OPEN" | "CLOSED" | "OUTSIDE" }
      } +
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getbinloc</name>
   <params>
      <bin_id>12004</bin_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin>
            <bin_id>12004</bin_id>
            <grid_id>1</grid_id>
            <xpos>21</xpos>
            <ypos>17</ypos>
            <depth>362</depth>
            <bin_mode>GRID</bin_mode>
         </bin>
      </bins>
   </params>
</response>
```

5.4.2.4 Error Codes

Error Code	Description

1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1049	Response overflow.
1055	Parameter min_bin_id is not valid (invalid format or out of range).
1056	Parameter max_bin_id is not valid (invalid format or out of range).

5.4.3 Get Bin State

The **getbinstate** method retrieves information about the Bins in the system. The method will be extended with additional response elements inside <bin>.

5.4.3.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only information about that specific Bin is returned.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only information about Bins with bin_id greater than or equal to min_bin_id is returned.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only information about Bins with bin_id less than or equal to max_bin_id is returned.

5.4.3.2 Response

bin_id:

Identifies the Bin.

content:

The Bin's content.

target_grid:

The preferred storage grid for the Bin.

xpos:

The physical X position of the Bin.

ypos:

The physical Y position of the Bin.

depth:

The physical depth in mm of the Bin.

port_id:

The port in which the Bin is located. (Numeric 0 if the Bin is not in a Port.)

bin_mode:

The current Bin mode, same codes as from LogPublisher.

5.4.3.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getbinstate" },
   element params {
      ( element bin_id { xsd:int { minInclusive = "1" } } ?,
         | ( element min_bin_id { xsd:int { minInclusive = "1" } } ?,
         element max_bin_id { xsd:int { minInclusive = "1" } } ?
         )
      ) ?
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element bins {
         element bin {
            element bin_id { xsd:int { minInclusive = "1" } },
            element content { xsd:short { minInclusive = "1" } },
            element target_grid { xsd:short { minInclusive = "1" } },
            element xpos { xsd:short { minInclusive = "1" } },
            element ypos { xsd:short { minInclusive = "1" } },
            element depth { xsd:short },
            element port_id { xsd:int {minInclusive = "1" } },
            element bin_mode { "G" | "T" | "P" | "R" | "O" | "C" |
               "X" | "F" }
         } +
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getbinstate</name>
   <params>
      <bin_id>10000</bin_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin>
            <bin_id>10000</bin_id>
            <content>1</content>
            <target_grid>1</target_grid>
            <xpos>34</xpos>
            <ypos>38</ypos>
            <depth>1124</depth>
            <port_id>0</port_id>
            <bin_mode>G</bin_mode>
         </bin>
      </bins>
   </params>
```

```
</response>
```

5.4.3.4 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1049	Response overflow.
1055	Parameter min_bin_id is not valid (invalid format or out of range).
1056	Parameter max_bin_id is not valid (invalid format or out of range).

5.4.4 Get Preparation Queue

The **getpreparequeue** method retrieves the preparation queue.

5.4.4.1 Parameters

None.

5.4.4.2 Response

bin_id:

Specifies a Bin.

5.4.4.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getpreparequeue" }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element bins {
         element bin_id { xsd:int { minInclusive = "1" } } *
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getpreparequeue</name>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin_id>12004</bin_id>
         <bin_id>25966</bin_id>
         <bin_id>53012</bin_id>
         <bin_id>18079</bin_id>
         <bin_id>24808</bin_id>
      </bins>
   </params>
</response>
```

5.4.4.4 Error Codes

Error Code	Description
1048	System is not available.
1049	Response overflow.

5.4.5 Get Port Queue

The **getportqueue** method retrieves the Port queue of the specified Port.

5.4.5.1 Parameters

port_id: (mandatory)

Identifies the Port.

5.4.5.2 Response

bin_id:

Specifies a Bin.

content:

Specifies a content code.

begingroup:

Starts a group in the Port queue.

endgroup:

Ends a group in the Port queue.

5.4.5.3 Message Format

Compact syntax (Request):

```
element methodcall {
```

```
      element name { "getportqueue" },
      element params {
         element port_id { xsd:short { minInclusive = "1" } }
      }
   }
```

Compact syntax (Response):

```
   element response {
      element params {
         element bins {
            ( element bin_id { xsd:int { minInclusive = "1" } }
            | element content { xsd:short { minInclusive = "0" } }
            | element begingroup { empty }
            | element endgroup { empty }
            ) +
         }
      }
   }
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getportqueue</name>
   <params>
      <port_id>2</port_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <bins>
         <bin_id>12004</bin_id>
         <bin_id>34434</bin_id>
         <bin_id>12088</bin_id>
      </bins>
   </params>
</response>
```

5.4.5.4 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1048	System is not available.
1049	Response overflow.

6. Port Processes

This chapter describes the standard port flow messaging, and special considerations for the different port types.

The process of integrating transfer cells is described in the chapter about Port Interface.

6.1 Standard Port Flows

This section includes example message flows of the standard port processes.

6.1.1 Putaway Process

In a typical putaway process, the WMS requests empty bins or bins with empty compartments of a specific size. The bin's content code normally reflects if the bin is empty, the compartment size, and the number of empty compartments.

With the Task Interface, the openport method includes the content codes that should be delivered during the session. AutoStore can deliver any of the specified content codes in any sequence.

```
WMS: openport( port_id=3, content=10, content=20 )
AS:  openport_response( ok )
```

With the Bin Interface, the WMS specifies the sequence of content codes to be delivered in the port queue. During the putaway process, the WMS appends content codes to the port queue.

```
WMS: openport( port_id=3 )
AS:  openport_response( ok )
WMS: appendportqueue( port_id=3, content=10, content=20,
content=10, ... );
AS:  appendportqueue_response( ok )
```

The following message flow is the same for Bin Interface and Task Interface:

The WMS requests the next bin:

```
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=13240 )
```

The response from AutoStore contains which bin that is selected.

After inventory is put in the bin, the WMS closes the bin with an updated content code, and requests another bin. The openbin/closebin process is repeated for each bin during the putaway process.

```
WMS: closenbin( port_id=3, bin_id=13240, content=11 )
AS:  closebin_response( ok )
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=18955 )
WMS: closenbin( port_id=3, bin_id=18955, content=21 )
AS:  closebin_response( ok )
...
```

When the port is no longer in use, the WMS shall close the port.

```
WMS: closeport( port_id=3 )
AS:  closeport_response( ok )
```

6.1.2 Picking Process with Task Interface

The ongoing process of creating and maintaining the task groups is not included in this description.

With the Task Interface, the openport method includes the category codes that should be delivered during the session.

```
WMS: openport( port_id=3, category=1 )
AS:  openport_response( ok )
```

The WMS requests the next bin:

```
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=23651, task_id=571233 )
```

The response from AutoStore contains which bin and task that is selected.

After the picking operation, the WMS closes the bin and requests another bin. The content code is updated because the compartment may become empty if the last item is picked.

The openbin/closebin process is repeated for each bin during the putaway process.

```
WMS: closebin( port_id=3, bin_id=23651, task_id=571233, content=11 )
AS:  closebin_response( ok )
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=38349, task_id=571234 )
WMS: closebin( port_id=3, bin_id=38349, task_id=571234, content=10 )
AS:  closebin_response( ok )
...
```

When the port is no longer in use, the WMS shall close the port.

```
WMS: closeport( port_id=3 )
AS:  closeport_response( ok )
```

6.1.3 Picking Process with Bin Interface

The ongoing process of preparing bins and maintaining the port queues is not included in this description.

With the Bin Interface, the openport method does not need additional parameters because the delivered bins are decided by the port queue.

```
WMS: openport( port_id=3 )
AS:  openport_response( ok )
```

The WMS requests the next bin:

```
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=23651)
```

The response from AutoStore contains which bin that is selected.

After the picking operation, the WMS closes the bin and requests another bin. The content code is updated because the compartment may become empty if the last item is picked.

The openbin/closebin process is repeated for each bin during the putaway process.

```
WMS: closenbin( port_id=3, bin_id=23651, content=11 )
AS:  closebin_response( ok )
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=38349 )
WMS: closenbin( port_id=3, bin_id=38349, content=10 )
AS:  closebin_response( ok )
...
```

When the port is no longer in use, the WMS shall close the port.

```
WMS: closeport( port_id=3 )
AS:  closeport_response( ok )
```

6.2 PickUpPort

> **NOTICE**
> **The PickUpPort requires April 2023 software release.**

The PickUpPort is typically used by customers to collect prepicked packages in a public zone. Bins are delivered into a drawer that can be manually opened and closed by the user. Bins are not delivered directly in front of the user. Instead, bins are delivered into the drawer and the users must manually open the drawer to access the bins. The port includes a locking mechanism that can lock the drawer. The port also includes a visual indication (light) of the state.

For most port types, the bin in mode OPEN is automatically delivered in front of the user. For the PickUpPort, the current bin (bin mode OPEN) is available for the user in the sense that it is delivered into the drawer and drawer is unlocked. The user must manually open the drawer to access the bin. The drawer can be manually closed and reopened as long as the bin mode remains OPEN. When the WMS logically closes the bin (*closebin*) the bin mode becomes CLOSED and the snap lock is enabled. The drawer may still be in the open position but when the drawer is closed the snap lock will prevent it from being reopened.

The WMS can use the method *getdrawerstate* to inspect the actual physical state of the drawer. The drawer state is also available from Log Publisher. The WMS may, for instance, check that the drawer is open before instructing the user to pick from it, and check that the drawer has been closed before finishing the session.

6.2.1 Get Drawer State

The **getdrawerstate** method returns the state of the drawer of the port.

The state includes the drawer position (closed or open) and the state of the snap lock (locked or unlocked).

6.2.1.1 Parameters

port_id: (mandatory)

Identifies the Port.

6.2.1.2 Response

drawer_position:

Returns the position of the drawer: UNKNOWN, CLOSED, OPEN.

snap_lock:

Returns the lock state of the drawer: UNKNOWN, ENABLED, DISABLED.

6.2.1.3 Message Format

Compact syntax (request):

```
element methodcall {
   element name { "getdrawerstate" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } }
   }
}
```

Compact syntax (response):

```
element response {
```

```
      element params {
         element drawer_position { xsd:string }
         element snap_lock { xsd:string },
         }
      }
   }
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getdrawerstate</name>
   <params>
      <port_id>2</port_id>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <drawer_position>OPEN</drawer_position>
         <snap_lock>DISABLED</snap_lock>
   </params>
</methodcall>
```

6.2.1.4 Error Codes

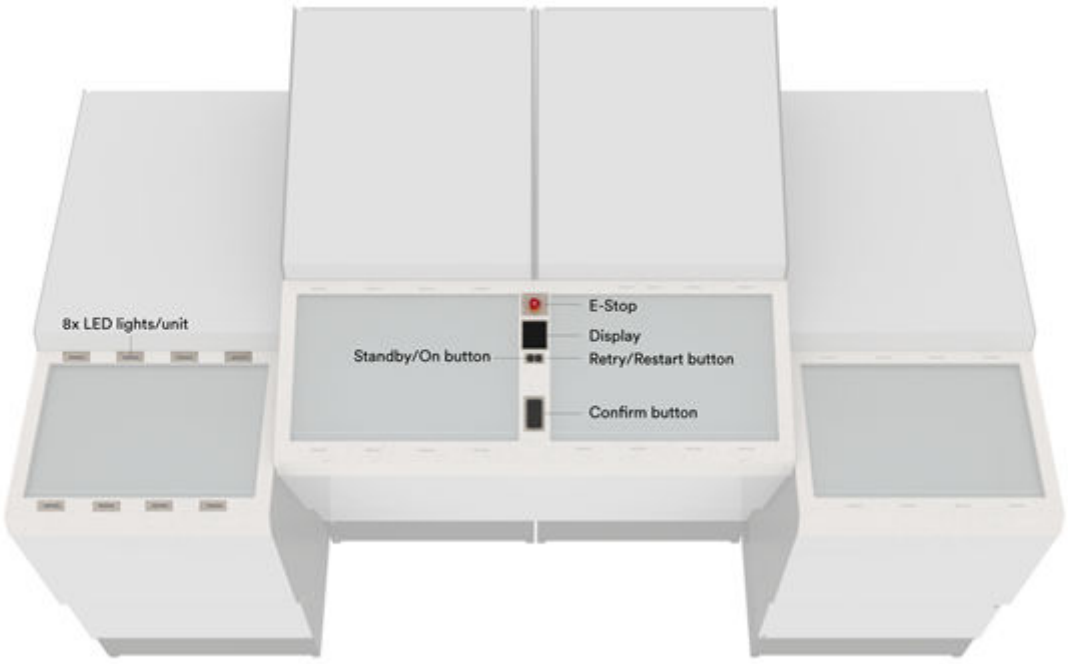
Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).

6.3 Fusion Port

NOTICE

The Fusion Port requires October 2023 software release.

6.3.1 Introduction



The Fusion Port One has four sub-modules. Each of these sub-modules may present bins independant of each other. In a typical pre-picking use case, the two locations in the middle are used as source locations to pick from. These two locations tilt the bin for better picking ergonomy. The two locations at the sides are typically used as target locations to pick into. These two locations do not tilt the bin.

The Fusion Port HMI includes:

- Pick-To-Light: Led strips indicate the source and target compartments
- Display: Shows the number of items to pick
- Confirm Button: A button that can be used to confirm the pick

The four sub-modules of the Fusion Port One are four logical Ports with separate Port IDs. The WMS controls the bin delivery to these ports like standard port processes. Each of these ports are controlled separately by the openport/openbin/closebin/closeport messaging.

The typical use case is to open the two source ports as standard picking processes, and the two target ports as standard put-away processes that request empty bins to store the pre-picked orders. One straightforward approach could be to always pick from the left source bin to the left target bin, and from the right source bin to the right target bin. But the WMS may implement more advanced batch picking strategies by controlling the bin flow to each individual port.

6.3.2 Pick-To-Light API

The Pick-To-Light API is optional. It is possible to operate the Fusion Port without using the Pick-To-Light API.

The Pick-To-Light API supports the visual indication to the operator: showing the number of items to pick on the display and lighting led strips that indicate the source compartment to pick from and the target compartment to pick into. The Log Publisher includes feedback when the operator pushes the confirm button for a specified pick operation.

The concept is based on Pick-To-Light commands (PTLs) created by the WMS. One PTL represents a picking operation and includes the required details (items to pick, source bin and compartment, target bin and compartment). The method *showpicktolight* creates a PTL and the method *hidepicktolight* deletes a PTL.

The PTL has a status that can be one of the following values:

Registered: PTL registered and waiting to be active.

Active: PTL active. Bins are open, display and led strips are active.

Confirmed: The PTL has been confirmed by pushing the confirm button.

The WMS may send multiple PTLs, but only one PTL per source port and target port combination. A PTL is not activated until both the source bin and the target bin are open in their respective ports. The target information is optional. For putaway operations or when picking into an external tote, we only need to specify the source information.

The confirm button confirms the active PTL (sets the status value = Confirmed) and turns off the leds. The confirm button has no effect if there are no active PTL. The confirm button does not automatically close any bins, it only changes the status of the PTL. The WMS must use *closebin* to close bins.

The Log Publisher segment PTL reflects updates and status changes to PTLs so it also informs when a PTL is activated (bins arrived in port) or completed (confirm button is pushed).

It is possible to use the visualization part of Pick-To-Light without using the confirm button. Closing a bin will automatically delete the active PTL (status = Active) that belongs to the port where the bin is closed. The WMS may use *hidepicktolight* instead of closing the bin if it wants to keep the bins in the port for more picks. Closing a port will delete all PTLs related to that port.

AutoStore will remember the PTLs in case of the WMS disconnects from the Log Publisher. A new connection to Log Publisher will receive the status of the existing PTLs, also including status=Confirmed. A new PTL for the same source port and target port combination will overwrite the old.

6.3.3 Show Pick-To-Light

The method **showpicktolight** requests visualization of a picking operation.

The method includes the visualization details: items to pick, source port, bin and compartment, target port, bin and compartment. The visualization request is also referred to as a Pick-To-Light command (PTL). The target information is optional and may be omitted or set to the value 0.

6.3.3.1 Parameters

source_port_id: (mandatory)

Identifies the source port.

target_port_id: (optional)

Identifies the target port.

source_bin_id: (mandatory)

Identifies the source bin.

target_bin_id: (optional)

Identifies the target bin.

source_compartment: (mandatory)

Identifies the source compartment.

target_compartment: (optional)

Identifies the target compartment.

quantity: (mandatory)

Number of items to pick.

6.3.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "showpicktolight" },
   element params {
      element source_port_id { xsd:int },
      element target_port_id { xsd:int },
      element source_bin_id { xsd:int },
      element target_bin_id { xsd:int },
      element source_compartment { xsd:int },
      element target_compartment { xsd:int },
      element quantity { xsd:int }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>showpicktolight</name>
   <params>
      <source_port_id>3</source_port_id>
      <target_port_id>4</target_port_id>
      <source_bin_id>15575</source_bin_id>
      <target_bin_id>20501</target_bin_id>
      <source_compartment>83</source_compartment>
      <target_compartment>11</target_compartment>
      <quantity>1</quantity>
   </params>
</methodcall>
```

6.3.3.3 Error Codes

Error Code	Description
1102	Parameter source_port_id is missing.
1103	Parameter source_port_id is not valid (invalid format or out of range).
1104	Parameter target_port_id is missing.
1105	Parameter target_port_id is not valid (invalid format or out of range).
1106	Parameter source_bin_id is missing.
1107	Parameter source_bin_id is not valid (invalid format or out of range).
1108	Parameter target_bin_id is missing.
1109	Parameter target_bin_id is not valid (invalid format or out of range).
1110	Parameter source_compartment is missing.
1111	Parameter source_compartment is not valid (invalid format or out of range).
1112	Parameter target_compartment is missing.
1113	Parameter target_compartment is not valid (invalid format or out of range).

1114	Missing parameter quantity.
1115	Parameter quantity is not valid (invalid format or out of range).
1116	Source port is not available.
1117	Target port is not available.
1118	Source bin does not exist.
1119	Target bin does not exist.

6.3.4 Hide Pick-To-Light

The method **hidepicktolight** cancels the visualization of a picking operation.

A visualization request is identified by the combination of a source port and a target port.

6.3.4.1 Parameters

source_port_id: (mandatory)

Identifies the source port.

target_port_id: (optional)

Identifies the target port.

6.3.4.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "hidepicktolight" },
   element params {
      element source_port_id { xsd:int },
      element target_port_id { xsd:int }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>hidepicktolight</name>
   <params>
      <source_port_id>3</source_port_id>
      <target_port_id>4</target_port_id>
   </params>
</methodcall>
```

6.3.4.3 Error Codes

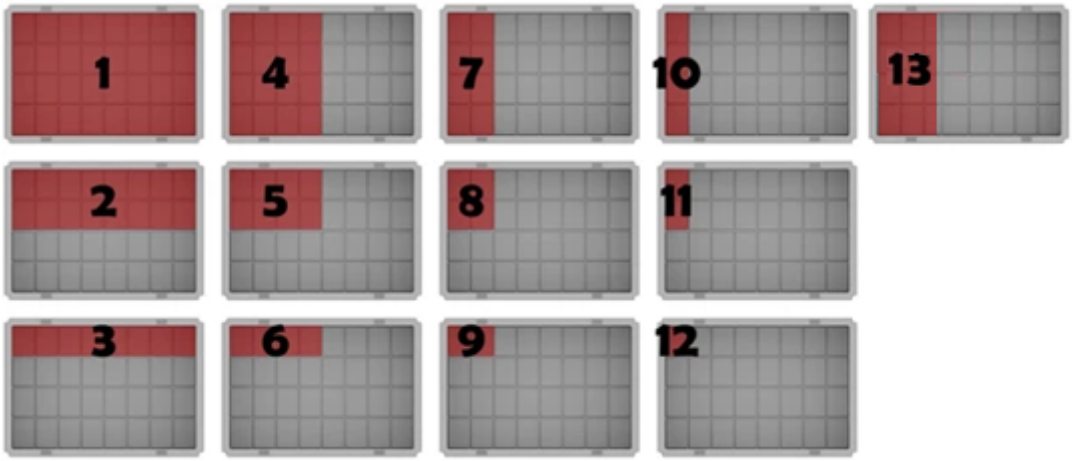
Error Code	Description
1102	Parameter source_port_id is missing.
1103	Parameter source_port_id is not valid (invalid format or out of range).
1104	Parameter target_port_id is missing.
1105	Parameter target_port_id is not valid (invalid format or out of range).

6.3.5 Compartment Definition

The Pick-To-Light API uses the compartment numbering described in this section.

The compartment type (bin configuration) is a code that identifies the size and shape of a compartment.

The following compartment types are defined:



The Compartment ID identifies both the size and shape of the compartment and the compartment's location inside the bin.

Compartment types 1-8 use a two-digit Compartment ID = *Type * 10 + Index*.

Compartment types 9-13 use Compartment ID = *Type * 100 + Index*.

The index starts at 1 and identifies the comparment's location inside the bin. Indexed from the NorthWest corner of the Bin, from left to right (west to east) for each row.

The Fusion Port supports the following compartment types: 1, 2, 4, 5, 7, 8, 13.

The Fusion Port manages the port's orientation (North vs. South).

6.3.6 Pick-To-Light Example

This example shows a simple message flow where we pick from the left source bin (port 2) to the left target bin (port 1), and from the right source bin (port 3) to the right target bin (port 4).

6.3.6.1 Message flow

Open two ports for picking and two ports for putaway:

```
WMS: openport( port_id=1, content=10 )
WMS: openport( port_id=2, category=1 )
WMS: openport( port_id=3, category=1 )
WMS: openport( port_id=4, content=10 )
```

Request the next bin in all ports:

```
WMS: openbin( port_id=2 )
AS:  openbin_response( bin_id=12345, task_id=100101 )
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=15575, task_id=100201 )
WMS: openbin( port_id=1 )
AS:  openbin_response( bin_id=20001 )
WMS: openbin( port_id=4 )
AS:  openbin_response( bin_id=20501 )
```

Send Pick-To-Light requests based on openbin responses:

```
WMS: showpicktolight( source_port_id=2, source_bin_id=12345,
source_compartment=51,
                      target_port_id=1, target_bin_id=20001,
target_compartment=11,
                      quantity=2 )
WMS: showpicktolight( source_port_id=3, source_bin_id=15575,
source_compartment=83,
                      target_port_id=4, target_bin_id=20501,
target_compartment=11,
                      quantity=1 )
```

AutoStore informs about which PTL that is activated (bins in mode OPEN) via the Log Publisher:

```
AS:  PTL,2,1,12345,20001,51,11,2,A
```

The WMS may show information about this picking operation in its GUI.

The user picks the items and confirms the pick with the Fusion Port confirm button. AutoStore informs about the confirmed picking operation via the Log Publisher:

```
AS:  PTL,2,1,12345,20001,51,11,2,C
```

The WMS registers the picking operation and requests the next source bin. (We assume there are more order lines to be picked into the same target bin, so the target bin is not closed.)

```
AS:  closebin( port_id=2, bin_id=12345, task_id=100101 )
WMS: openbin( port_id=2 )
AS:  openbin_response( bin_id=18536, task_id=100102 )
WMS: showpicktolight( source_port_id=2, source_bin_id=18536,
source_compartment=11,
                      target_port_id=1, target_bin_id=20001,
target_compartment=11,
                      quantity=2 )
```

Then the next PTL is ready and activated. If the next bins are available, this may happen straight after the confirmation of the previous pick (before the above closebin/openbin).

```
AS:  PTL,3,4,15575,20501,83,11,1,A
```

And picked and confirmed:

```
AS:  PTL,3,4,15575,20501,83,11,1,C
```

A new source bin is requested:

```
AS:  closebin( port_id=3, bin_id=15575, task_id=100201 )
WMS: openbin( port_id=3 )
AS:  openbin_response( bin_id=19244, task_id=100202 )
```

If the target bin is full (or if it was the last order line in that order), a new target bin can be requested:

```
AS:  closebin( port_id=4, bin_id=20501 )
WMS: openbin( port_id=4 )
AS:  openbin_response( bin_id=14681 )
WMS: showpicktolight( source_port_id=3, source_bin_id=19244,
source_compartment=84,
                      target_port_id=4, target_bin_id=14681,
target_compartment=11,
                      quantity=3 )
```

And the picking process can continue with this type of message flow during the picking session.

Log Publisher also reports PTLs when they are registered (status=R), but these segments are not included in the example.

6.3.6.2 Alternative flow

In the example, we created the PTLs for both source ports based on the openbin responses. This enables the Fusion Port to activate the next PTL based on which bins that are first available in the Fusion Port.

Instead of sending multiple PTLs, the WMS can check itself when bins are put into the port. Then, when both source and target bins are available, the WMS sends the PTL.

```
AS:  BL,12345,1,xpos,ypos,depth,P
AS:  BL,20001,1,xpos,ypos,depth,P
WMS: showpicktolight( ... )
```

To avoid waiting in port, the WMS should either send all the alternative PTLs or check itself when bins are available. If the WMS just decides to pick from one of the ports first regardless of which bins that are available, time may be wasted on waiting in the port.

6.4 MultiPort configuration of the Fusion Port

6.4.1 Introduction

In the default configuration of a Fusion Port (or Fusion Port One), all sub-modules are logical ports with separate port IDs.

In some use cases, the WMS may prefer to view the two source port modules as one logical port. For instance if the WMS wants to use both modules of a Fusion Port for picking, but only want to pick one task group at the time.

It is possible to reconfigure a Fusion Port (or Fusion Port One) so that the two source modules appears as one logical port. In addition to the sub-modules, each Fusion Port also has a logical port with its own Port ID of port type MULTI_PORT. In the default setup, this port cannot be usedis (port mode shows as DISABLED in ASConsole). The methods *enable_multiport* and *disable_multiport* can be used to reconfigure a Fusion Port. When the multiport is enabled, the two logical source ports cannot be used (port mode shows as MULTIPORT_PUPPET in ASConsole).

From a WMS perspective, the multiport behaves like one standard port. It is controlled via the API like any other port by using the standard port methods.

A port can only have one open bin at the same time. Therefore, only one of the source ports will have a bin available (lid open) at the same time. Typically the closebin/openbin between each bin will typically close lid on one side and open the lid on the other side.

Enabling the multiport of a Fusion Port One only affects the source port modules. The two target ports are still handled as two separate ports.

6.4.2 Enable MultiPort

The **enable_multiport** method enables the specified multiport and disables the individual ports that make up the multiport.

6.4.2.1 Parameters

port_id: (mandatory)

Identifies the MultiPort.

6.4.2.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "enable_multiport" },
   element params {
      element port_id { xsd:int }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>enable_multiport</name>
   <params>
      <port_id>5</port_id>
   </params>
</methodcall>
```

6.4.2.3 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).

6.4.3 Disable MultiPort

The **disable_multiport** method disables the specified multiport and enables the individual ports that make up the multiport.

6.4.3.1 Parameters

port_id: (mandatory)

Identifies the MultiPort.

6.4.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "disable_multiport" },
   element params {
      element port_id { xsd:int }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>disable_multiport</name>
   <params>
      <port_id>5</port_id>
   </params>
</methodcall>
```

6.4.3.3 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).

6.5 Miscellaneous Port Methods

6.5.1 Configure Port

The **configureport** method updates the configuration of the specified Port.

The following settings can be updated:

Configuration Name	Configuration Value	
weight_config	0	Overload always disabled.
	1	Overload always enabled.
	2	(Deprecated) Overload disabled in picking mode.
	3	No weight.
	4	Overload always enabled. Silent mode. The local port will not signal or trigger error on overload.
enableweight	True	Enable the weight in the Port.

(Deprecated)		(same as weight_config = 1)
	False	Disable the weight in the Port. (same as weight_config = 0)
robotqueue	Integer value that specifies how many bins should be queued towards the port. See section Port Configurations (on page 147).	
changeportbinqueue (Deprecated)	Same as *robotqueue*.	
nextTaskGroupBinLookahead	Integer value that specifies the number of bins the system will use to determine when to notify about upcoming task groups for each port. This configuration is only available for the Task Interface.	

6.5.1.1 Parameters

port_id: (mandatory)

Identifies the Port.

configname: (mandatory)

The name of the configuration that should be updated.

configvalue: (mandatory)

The new value of the specified configuration.

6.5.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "configureport" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element configname { xsd:string },
      element configvalue { xsd:string }
   }
}
```

Request example (enableweight configuration):

```
<?xml version="1.0"?>
<methodcall>
   <name>configureport</name>
   <params>
      <port_id>3</port_id>
      <configname>enableweight</configname>
      <configvalue>True</configvalue>
   </params>
</methodcall>
```

Request example (changeportbinqueue configuration):

```
<?xml version="1.0"?>
<methodcall>
   <name>configureport</name>
   <params>
```

```
        <port_id>3</port_id>
        <configname>changeportbinqueue</configname>
        <configvalue>2</configvalue>
      </params>
   </methodcall>
```

6.5.1.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1048	System is not available.
1084	Invalid parameter configname.
1085	Invalid parameter configvalue.

6.5.2 Get Port Configuration

The method **getportconfig** returns information about the specified Port.

6.5.2.1 Parameters

port_id: (mandatory)

Identifies the Port.

6.5.2.2 Response

weight_equipment:

0 - No weight installed
1 - Overload only installed
2 - Scale weight installed

weight_config:

0 - Overload always disabled
1 - Overload always enabled
2 - Overload in pick mode enabled (Deprecated)
3 - No weight
4 - Overload always enabled. Silent mode (port will not signal or trigger error on overload)

6.5.2.3 Message Format

Compact syntax (Request):

```
   element methodcall {
```

```
      element name { "getportconfig" },
      element params {
         element port_id { xsd:int { minInclusive = "1" } }
      }
   }
```

Compact syntax (Response):

```
   element response {
      element params {
         element weight_equipment { xsd:int },
         element weight_config { xsd:int }
      }
   }
```

Request example:

```
   <?xml version="1.0"?>
   <methodcall>
      <name>getportconfig</name>
      <params>
         <port_id>4</port_id>
      </params>
   </methodcall>
```

Response example:

```
   <?xml version="1.0"?>
   <response>
      <params>
         <weight_equipment>2</weight_equipment>
         <weight_config>1</weight_config>
      </params>
   </response>
```

6.5.2.4 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).

6.5.3 Get Port Weight

The **getportweight** method retrieves weight information from the specified Port.

> **NOTICE**
> **This description applies to the April 2023 software release.**

6.5.3.1 Parameters

port_id (mandatory):

Identifies the Port.

timeout_mode:

Determines if one should wait for precision. Can be set to "L" (for Low/unstable precision), "H" (for High/stable precision) or not set at all if one should not wait for precision.

timeout_value:

Specifies the timeout in milliseconds. If not defined, the default value for timeout_value is 0 milliseconds.

6.5.3.2 Response

weight_status:

Determines if the weight is set with "L" (Low precision), "H" (High precision), "W" (Waiting for precision), or “NA” (Not available: weight_value is locked to 0 gram).

weight_value:

The weight in grams.

weight_overload:

Determines if the Bin is overloaded (1) or if the Bin is not overloaded (0).

6.5.3.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getportweight" },
   element params {
      element port_id { xsd:int { minInclusive = "1" } },
      element timeout_mode { "L" | "H" } ?,
      element timeout_value { xsd:int } ?
   }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element portweight {
         element weight_status { "L" | "H" | "W" | "NA },
         element weight_value { xsd:int },
         element weight_overload { 1 | 0 }
      }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>getportweight</name>
   <params>
      <port_id>4</port_id>
      <timeout_mode>L</timeout_mode>
      <timeout_value>10</timeout_value>
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
```

```
<response>
   <params>
      <portweight>
         <weight_status>L</weight_status>
         <weight_value>370</weight_value>
         <weight_overload>0</weight_overload>
      </portweight>
   </params>
</response>
```

6.5.3.4 Error Codes

Error Code	Description
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1092	Invalid parameter timeout_mode.
1093	Invalid parameter timeout_value.

6.5.4 Flush Port Bin

The **flushportbin** method requests AutoStore to move the specified Bin from a closed position of the port into the grid.

6.5.4.1 Parameters

port_id: (mandatory)

Identifies the Port where the bin is.

bin_id: (mandatory)

Identifies the Bin that should be moved.

6.5.4.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "flushportbin" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>flushportbin</name>
```

```
    <params>
       <port_id>5</port_id>
       <bin_id>14861</bin_id>
    </params>
</methodcall>
```

6.5.4.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1002	The specified Port is already open.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1024	Mandatory parameter bin_id is missing.
1072	The specified position does not belong to the specified Port.
1079	Illegal Bin mode.
1087	Illegal Port type.

7. Forecast Bin Preparation

The forecast concept enables the WMS to request low priority Bin preparation. The concept is supported by both the Bin Interface and the Task Interface.

7.1 Description

The forecasting concept comes in addition to existing functionality. Existing concepts (such as preparation queue and task groups) are not changed.

Forecasting adds the possibility of low priority preparation of Bins. Forecast Bins will be prepared when there is free capacity. Forecast Bins will not occupy available Transit/Prepare positions. Typically, forecast Bins will float to a higher level (not necessarily the top level).

7.1.1 Bin Mode Forecast

The Bin mode FORECAST behaves like bin mode GRID, except that the Bin will be moved towards the upper levels based on available digging capacity.

A Bin will leave the FORECAST mode if it changes to another mode, for example if the Bin is put into a Port, or if the Bin has been prepared (due to the preparation queue), or if the forecast mode is canceled (using the *cancelforecastbin* method).

A Bin is never defined as 'finished forecast prepared'. The current position of the Bin (and its Bin mode) is continuously available from the Log Publisher module.

7.2 Forecast Methods

7.2.1 Forecast Bin

The **forecastbin** method requests one or multiple Bins to be changed to bin mode *forecast*.

Even though it is possible to forecast as many Bins as one would like, no more than 1000 bins can be forecast in a single **forecastbin** message.

7.2.1.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only that specific Bin will be requested to be changed to mode *forecast*.

It is possible to provide multiple bin_id elements, in order to request an array of Bins to be changed to mode *forecast*. The maximum number of Bins to forecast in one array is 1000.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only Bins with bin_id greater than or equal to min_bin_id, and bin_id less than or equal to max_bin_id, are requested to be changed to mode *forecast*.

The maximum number of Bins to be forecast in one message is 1000, meaning that the difference between min_bin_id and max_bin_id can be no more than 1000.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only Bins with bin_id less than or equal to max_bin_id, and bin_id greater than or equal to min_bin_id, are requested to be changed to mode *forecast*.

The maximum number of Bins to be forecast in one message is 1000, meaning that the difference between max_bin_id and min_bin_id can be no more than 1000.

7.2.1.2 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "forecastbin" },
   element params {
      ( element bin_id { xsd:int { minInclusive = "1" } } ?,
         | ( element min_bin_id { xsd:int { minInclusive = "1" } } ?,
         element max_bin_id { xsd:int { minInclusive = "1" } } ?
         )
      )
   }
}
```

Request example (Single bin):

```
<?xml version="1.0"?>
<methodcall>
   <name>forecastbin</name>
   <params>
      <bin_id>14861</bin_id>
   </params>
</methodcall>
```

Request example (Array of bins):

```
<?xml version="1.0"?>
<methodcall>
   <name>forecastbin</name>
   <params>
      <bin_id>14861</bin_id>
      <bin_id>14863</bin_id>
      <bin_id>14864</bin_id>
   </params>
</methodcall>
```

Request example (Range of bins):

```
<?xml version="1.0"?>
<methodcall>
   <name>forecastbin</name>
   <params>
      <min_bin_id>14861</min_bin_id>
      <max_bin_id>14864</max_bin_id>
   </params>
</methodcall>
```

7.2.1.3 Error Codes

Error Code	Description

1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1024	Mandatory parameter bin_id is missing.
1049	Response overflow.
1078	Bin is in Port.
1079	Illegal Bin mode.
1080	Method not supported.

7.2.2 Cancel Forecast Bin

The **cancelforecastbin** method requests one or multiple Bins to leave bin mode *forecast*.

Even though the number of Bins with bin mode *forecast* can be as high as one would like, it is not possible to cancel the forecast of more than 1000 Bins in a single **cancelforecastbin** message.

7.2.2.1 Parameters

bin_id: (optional)

If the parameter bin_id is provided, only that specific Bin will be requested to leave mode *forecast*.

It is possible to provide multiple bin_id elements, in order to request an array of Bins to leave mode *forecast*. The maximum number of Bins that can be requested to leave mode *forecast* in one array is 1000.

min_bin_id: (optional)

If the parameter min_bin_id is provided, only Bins with bin_id greater than or equal to min_bin_id, and bin_id less than or equal to max_bin_id, are requested to leave mode *forecast*.

The maximum number of Bins that can be requested to leave mode *forecast* is 1000, meaning that the difference between min_bin_id and max_bin_id can be no more than 1000.

max_bin_id: (optional)

If the parameter max_bin_id is provided, only Bins with bin_id less than or equal to max_bin_id, and bin_id greater than or equal to min_bin_id, are requested to leave mode *forecast*.

The maximum number of Bins that can be requested to leave mode *forecast* is 1000, meaning that the difference between max_bin_id and min_bin_id can be no more than 1000.

7.2.2.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "cancelforecastbin" },
   element params {
      ( element bin_id { xsd:int { minInclusive = "1" } } ?,
         | ( element min_bin_id { xsd:int { minInclusive = "1" } } ?,
         element max_bin_id { xsd:int { minInclusive = "1" } } ?
         )
      )
   }
}
```

Request example (Single bin):

```
<?xml version="1.0"?>
<methodcall>
   <name>cancelforecastbin</name>
   <params>
      <bin_id>14861</bin_id>
   </params>
</methodcall>
```

Request example (Array of bins):

```
<?xml version="1.0"?>
<methodcall>
   <name>cancelforecastbin</name>
   <params>
      <bin_id>14861</bin_id>
      <bin_id>14863</bin_id>
      <bin_id>14864</bin_id>
   </params>
</methodcall>
```

Request example (Range of bins):

```
<?xml version="1.0"?>
<methodcall>
   <name>cancelforecastbin</name>
   <params>
      <min_bin_id>14861</min_bin_id>
      <max_bin_id>14864</max_bin_id>
   </params>
</methodcall>
```

7.2.2.3 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1024	Mandatory parameter bin_id is missing.
1049	Response overflow.
1078	Bin is in Port.
1079	Illegal Bin mode.
1080	Method not supported.

7.3 Guidelines

The recommended maximum number of Bins to forecast is 25% of the total number of Bins in the system. This maximum number (unique Bins) should include all kinds of preparation requests (forecasting, task groups, preparation queue, Port queues).

In a scenario with nightly forecast preparation, it is recommended to only forecast the Bins that are needed next day. If they are needed on a later date, then the forecast should be delayed until the night before they are needed.

It is possible to forecast at any time. Forecasting will mainly use spare Robot capacity.

Pay attention to the precision of the forecast. Forecast Bins that are not needed will have a negative impact (Bins floating up displace other Bins).

8. Controlling Bin Locations

This chapter describes the concepts and features for controlling the physical Bin locations.

NOTICE

Avoid using the features marked as Deprecated. Features marked as deprecated are not supported by AS Router.

NOTICE

AS Router only supports one code per cell. Use ranges when multiple codes are needed.

8.1 Introduction

The features and concepts described in this chapter make it possible to assign specific grid areas to store specific bins or SKUs. One use case is to assign one area of the grid for ESD bins and another area for non-ESD bins.

These features also enable installations to operate without bin preparation or digging for the normal processes. This is supported by not mixing different SKUs in the same stack, so that outgoing bins can always be taken from the top. Please note that such use cases will typically cause a lower fill rate in the grid as many stacks will be partially filled.

8.1.1 Bin Content Code

Every AutoStore Bin is assigned a content code. Previously, the content codes specified whether a Bin is empty or not (and possibly also compartment information). They were used for the put-away process. The WMS may request bins (in the port queue) by specifying content codes instead of specific Bin IDs.

The same Bin content code is central in *Controlling the Location of Bins*, but depending on the use case, the Bin content code may also reflect the inventory of the Bin (SKU or batch).

The Bin content code is controlled by the WMS, and is updated via the interface. The mapping between SKU/batch and content codes is defined by the WMS. The Bin content code may identify a SKU or a group of SKUs. The WMS may also use multiple content codes per SKU to control the bin flow.

8.1.2 Cell Content Code

Each cell (stack) in the grid can be associated with one or multiple cell content codes.

In cells that have cell content code(s), AutoStore will only store bins that have Bin content code equal to one of the cell's cell content codes.

When assigning cell content codes to a cell, the following applies:

- Cell content code is the same datatype as the Bin content code (integer, range [1-32767]).
- Content code ranges may be used in cases where multiple content codes per cell is needed.
- AS Router supports a single value per cell. A content code or a content range.
- AS Planner supports up to 8 cell content codes in a cell.

Empty cells without content code are available for bins with any Bin content code (except the case when this is restricted by the *OverflowToFreeCell* configuration).

The cell content code is controlled by the WMS and can be updated via the interface during operation. The cell content code can also be configured by using the AutoStore Console.

8.1.3 Content Code Ranges

A *Content Code Range* defines a group that incudes a specified range of content codes.

This group can be used as a cell's cell content code and addresses the limitation that a cell only can have up to 8 cell content codes.

Content code ranges cannot be used instead of content codes in port queues or in openport parameters.

Content code ranges are defined by using AutoStore Console.

8.1.4 Interface

The functionality for *Controlling the Location of Bins* is available in both Bin Interface and Task Interface.

Much of the behavior is controlled by static configurations.

The Bin content code is set by using the interface methods: *closebin*, *insertbin*, and *updatebin*.

A new method is introduced: *updatecellandbincontent*. This method updates the content code of a specified cell and all the Bins that are stored in that cell. (This method is often not required; it is only needed for some specific use cases.) Using the AutoStore Console, it will be possible to update the cell content code without changing the content code of the Bins that are stored in the cell.

> **NOTICE**
> **See section Feature Compability about feature support in ASPlanner and ASRouter.**

8.2 Configurations

8.2.1 Main Configurations

AutoStore includes the following main configurations to control the behavior:

Configuration	Value	Description
OverflowToFreeCells	True or False	A Bin cannot be placed in a cell without content code when there exist cells that have a cell content code equal to the given Bin's content code, unless this setting is enabled. In other words: A Bin can be placed in an empty stack without cell content code if, and only if, either: (a) There are no cells with cell content code equal to the Bin's content code. (b) Configuration *OverflowToFreeCells==True*.
UniqueContentInStack	True or False	*UniqueContentInStack feature is deprecated.* When this setting is enabled, different Bin content codes cannot be mixed in the same stack.

StackFifoTimeStamp	True or False	*StackFifoTimeStamp feature is deprecated.* When this setting is enabled, incoming Bins will be placed in locations according to the timestamp of the cell (newest cell timestamp), and outgoing Bins are selected based on the timestamp of each cell (oldest cell timestamp). This configuration can only be True if *UniqueContentInStack* is also enabled.
ContentRandomTarget	True or False	*ContentRandomTarget feature is deprecated.* This setting affects the location where inserted Bins are placed. If several options remain after searching for the target with lowest consumer distance, the selection can be random or by shortest driving distance. Random will spread the content throughout the Grid, beneficial if it is inserted at one location, but used at multiple. Shortest driving distance will make it greedy, choosing the option that is most effective at the actual moment in time.

8.2.1.1 Terms

The features *UniqueContentInStack* and *StackFifoTimeStamp* that support these terms are deprecated.

STACKFIFO means that *UniqueContentInStack*=True and *StackFifoTimeStamp*=True.

In this mode, Bins with different content code cannot be stored in the same stack.

Each cell includes a time stamp that is set when the first Bin is stored (in an empty stack), and cleared when the stack becomes empty. When Bins are inserted, they will be stored in the newest stack that is used by that content code. Hence, the stacks are filled one by one. When a new empty stack is needed, AutoStore will dynamically select one (according to cell content restrictions and content consumer configurations). When Bins are taken out, they will be taken from the oldest stack that is used by the content code. Stacks are emptied one by one – by FIFO based on the cells' time stamp.

Empty Bins are treated specially by AutoStore. A specific content code must be reserved for empty Bins. Empty Bins do not follow FIFO.

BATCHFIFO means that *UniqueContentInStack*=True and *StackFifoTimeStamp*=False.

This mode is similar to STACKFIFO, but does not use the cell time stamp. Typically, the WMS assigns a content code to each batch (instead of each SKU), and in the picking process the WMS requests a specific batch to be used. This gives AutoStore more flexibility, because there is no FIFO requirements between the stacks that belong to the same batch.

8.2.2 Content Consumers

The Content Consumers feature is deprecated.

The Content Consumers configuration specifies where (which ports) the different Bin content codes are expected to be used (consumed).

Content code ranges can be used in the configuration. A content code range can be associated with a list of Ports (consumers) where the Bins are expected to be delivered. The configuration may contain many such associations. The same content code may only be included in one association (no overlapping ranges).

AutoStore will use the Content Consumers configuration when it selects among legal storage location in the Grid for a Bin.

The Content Consumer configuration does not control which Ports the Bins are delivered to. The Content Consumer configuration only tells AutoStore where the recommended storage locations in the Grid are. The WMS can request the Bins in any Port independent of the Content Consumer configurations. For best performance, they should be requested in Ports close to their current position.

A Bin is considered being placed in the *Proximity* of its configured consumers if it is placed among the 50 nearest cells of any of its consumers. Nearest means shortest Manhattan traveling distance (L-shape route).

8.3 Interface Methods

8.3.1 Update Cell and Bin Content

The **updatecellandbincontent** method updates the content code of a specified cell and the Bins that are stored in that cell.

Normally, this method is not required.

The purpose of this method is to support scenarios where the WMS provides the "StackFifo" logic. The WMS can use three content codes (incoming, storage, outgoing) per SKU. Bins are inserted with the content code for incoming, and selected to the Port with the content code for outgoing. The WMS then uses this method to change the mode of each stack.

8.3.1.1 Parameters

x: (mandatory)

Specifies the X position.

y: (mandatory)

Specifies the Y position.

content: (mandatory)

Specifies new content code.

8.3.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "updatecellandbincontent" },
   element params {
      element x { xsd:short { minInclusive = "1" } },
      element y { xsd:short { minInclusive = "1" } },
      element content { xsd:short { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>updatecellandbincontent</name>
   <params>
      <x>41</x>
      <y>17</y>
      <content>535</content>
   </params>
```

```
</methodcall>
```

8.3.1.3 Error Codes

Error Code	Description
1006	Parameter content is not valid (invalid format or out of range).
1066	Mandatory parameter x is missing.
1067	Parameter x is not valid (invalid format or out of range).
1068	Mandatory parameter y is missing.
1069	Parameter y is not valid (invalid format or out of range).

8.4 Managing Frozen Zone

In a solution that supports frozen and chilled temperature storage areas in the same grid, it is critical that the bins are stored in the correct area. This is controlled by using content codes.

NOTICE

Installations with different temperature zones in the same Grid require AS Router. Although AS Planner supports cell content codes, AS Planner will to a greater extent allow bins to temporarlily be in another zone.

8.4.1 Defining the Zones

Normally, different content codes are used to reflect if the bins are empty or not, and the number of available compartments in the bins. In a multi-temperature environment, the content code must also reflect which zone the bin belongs. Each temperature zone shall be assigned a range of content codes. The Content Ranges shall not overlap.

- Each temperature zone shall have it's specific Content Code Range.
- Each grid storage cell shall be defined with Cell Content = its zone's Content Code Range.
- Every bin shall belong to one and only one zone and use content codes that belongs to its zone's Content Code Range.

It is the responsibility of the WMS to maintain a correct bin content code. Assigning wrong content code to a bin may cause the bin to be stored in the wrong temperature zone and may damage the bin's content.

8.4.2 Controlling Bin time outside its Zone

In order to preserve the quality of the stored inventory, the WMS must control the time a bin spends outside its storage zone. The WMS should monitor that the handling time in the port is compatible with the inventory of the bin, and also consider total time outside the zone within a time interval if the bin is used several times.

To avoid that a bin is stuck in a port because the operator has left the port, the WMS should implement some kind of timeout and automatical log out. If the port is not being used, it is important to close the bin and close the port so that the bin can be returned to its storage zone.

To avoid that bins stay too long in the port, AutoStore has introduced a port configuration: *AutoFlush.* When AutoFlush is enabled, the system will, after a certain period with idle time (time since the port had a selected bin), automatically flush the bins in the port so that they are returned to their storage zone. The feature does not flush the bins with bin mode OPEN. Bins that are open are controlled by the WMS and should be handled by some timeout functionality in the WMS.

AutoFlush can be configured for each port, and can have a different timeout value for each port. AutoFlush can be configured via AS Console, and should normally be set in the initial setup. It is important that AutoFlush is enabled in multi-temperature solutions.

9. Port Interface

The Port Interface controls the physical exchange of Bins between AutoStore and external Port implementations. The Port Interface can be used with the Bin Interface as well as with the Task Interface. The current version of the Port Interface only supports single cell Ports.

The Ports that are controlled by the Port Interface must be configured as port type **CUSTOM**.

9.1 Port Process

9.1.1 Allocate Cell

In order to get Bins delivered to the Port, the Port must allocate cells for Bin reception. The purpose of the cell allocation is to let AutoStore plan the flow of Bins to the Port. Cells are normally allocated before the Port is ready to physically receive the Bin.

One cell can only have one pending allocation. A new allocation request will override the previous one. The WMS normally sends the next allocation request immediately after AutoStore has confirmed the Bin operation (*bintoport*/ *binfromport*).

(If a Port includes multiple port cells, the Port must allocate the cells in the same sequence as that in which it wants the Bins delivered. However, the Port Interface does not support Ports with multiple port cells.)

Normally, when a Bin is delivered to a Port another Bin is returned from the Port. However, if the Port uses separate cells for returning the Bins to the Grid, the Port needs to allocate these cells as well. The purpose of such announcements is to let AutoStore plan the flow of empty Robots to the Port. The two allocation sequences (delivery and fetching) are independent of each other.

The Port allocates cells for interaction with AutoStore by sending the *portcellallocate* message.

9.1.2 Exchange Bins to and from Grid

Cell allocation alone does not permit AutoStore to physically use a cell.

The Port allows AutoStore to physically deliver a Bin to a cell by sending the *portlocationready* message. The Port allows AutoStore to physically retrieve a Bin from a cell by sending the *portbinready* message.

The *portlocationready* message indicates whether AutoStore should just deliver a Bin, or if it should also retrieve a returned Bin (Bin exchange).

AutoStore confirms the physical operations in a cell via the Log Publisher Interface.

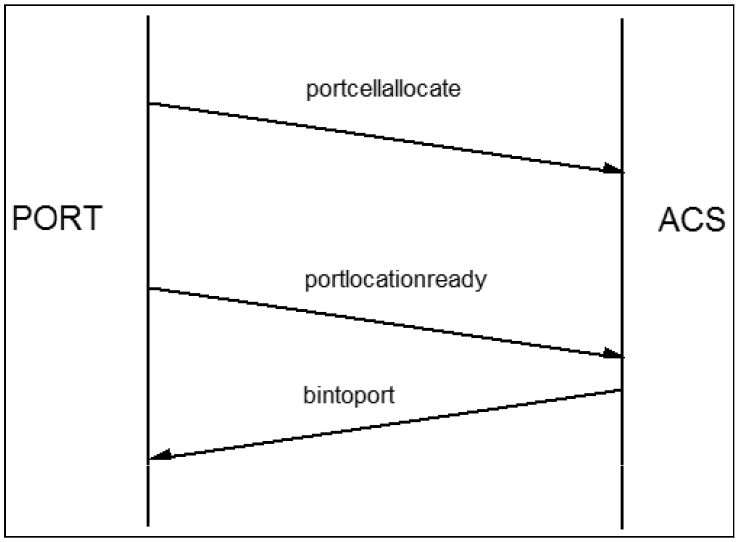
The logical message *bintoport* is sent as an ordinary BL segment (new Bin location) in the Port's position. It confirms that a Bin has been delivered to a cell, and AutoStore relinquishes control of that cell.

The logical message *binfromport* is sent as an ordinary BL segment (Bin on Robot or in another position) of the Bin specified by *portbinready*. It confirms that a Bin has been retrieved from a cell, and AutoStore relinquishes control of the cell.

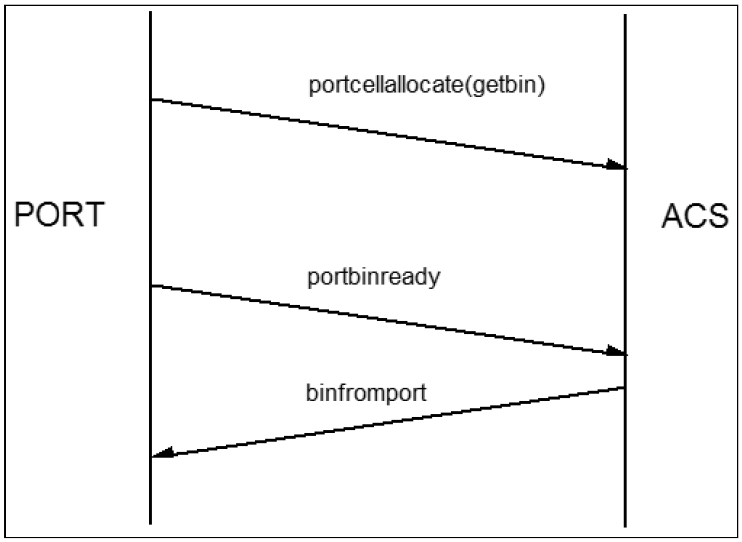
Please note that the BL segment that reflects the *binfromport* may specify a Grid position that is different from the Port. The BL segment may confirm the Bin on a robot (no position), but it may also directly specify the next position without first sending a BL without position. Both these cases should be interpreted as *binfromport*.

9.2 Message Flow

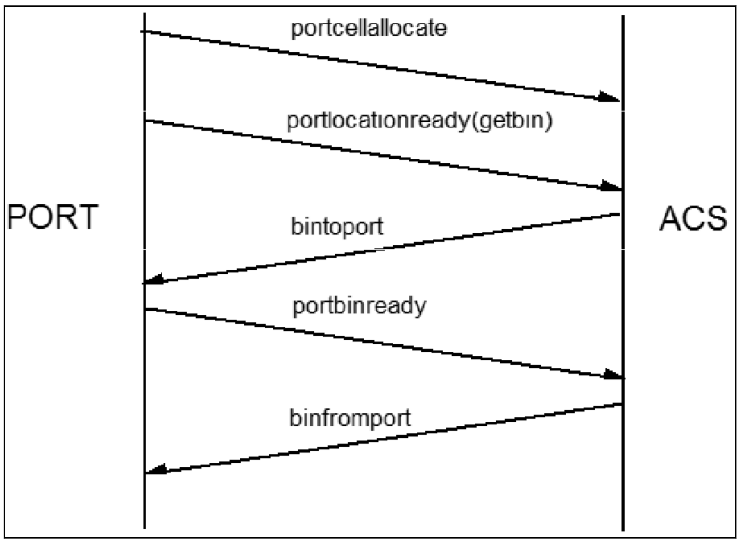
9.2.1 Port Retrieve Bin from Grid



9.2.2 Port Deliver Bin to Grid



9.2.3 Port Exchange Bin with Grid



9.3 Examples

9.3.1 Port with Outgoing Bins to Conveyor Belt

We assume that a process of preparing Bins and planning Bins for the Ports is active. This process may cause the following communication:

```
WMS: preparebin ( 15001, 15002, 15221, 15227, 15238, 15287, 16802 )
LPI:  BM,15002,T
LPI:  BM,15238,T
LPI:  BM,15001,T
WMS: appendportqueue( port_id=3, bin_id=15001, 15002, 15238 )
```

The process should continuously add new Bins to the "prepare" queue and the port queues during operation. If the Task Interface is used, the process should continuously add task groups instead. The rest of this example applies to both Task Interface and Bin Interface.

A Port, (Port number 3) located at position (15,12), handling incoming Bins going out to an external belt system can cause the following communication:

```
WMS: openport( port_id=3 )
```

The Port is activated. Default mode is **OPEN.**

```
WMS: portcellallocate( port_id=3, x=15, y=12 )
```

The WMS requests a Robot to the Port.

```
WMS: openbin( port_id=3 )
WMS: portlocationready( port_id=3, x=15, y=12 )
```

The WMS confirms that the position (15,12) is ready , and relinquishes control of the cell.

```
LPI:  BL,15001,1,15,12,1,P
```

AutoStore confirms via the Log Publisher Interface that Bin 15001 is put in the cell, and relinquishes control of the cell. Note that the depth should be ignored in the Port cell.

```
WMS: removebin( port_id=3, bin_id=15001 )
```

The WMS confirms that the Bin is removed from the Port.

```
WMS: portcellallocate( port_id=3, x=15, y=12 )
```

The WMS requests a Robot to the Port.

```
WMS: openbin( port_id=3 )
WMS: portlocationready( port_id=3, x=15, y=12 )
```

The WMS confirms that the position (15,12) is ready, and relinquishes control of the cell.

```
LPI:  BL,15002,1,15,12,1,P
```

AutoStore confirms via the Log Publisher Interface that Bin 15002 is put in the cell, and relinquishes control of the cell.

```
WMS: removebin( port_id=3, bin_id=15002 )
```

The WMS confirms that the Bin is removed from the Port.

The process is repeated as long as the Port is active.

```
WMS: closeport( port_id=3 )
```

When the Port is no longer in use, the WMS should use closeport to release the resources that AutoStore plans for the open Ports.

9.3.2 Port with Outgoing Bins to Conveyor Belt (mode AUTO)

This example illustrates the use of Port mode AUTO. If this mode is used, the WMS should be especially aware of the fact that Bins are automatically set to OUTSIDE mode after they are delivered to the Port. The WMS must be able to allow for the fact that bins may be delivered and enter OUTSIDE mode while the Log Publisher connection is closed. (It is recommended to use mode OPEN instead of mode AUTO.) Port mode AUTO cannot be used with the Task Interface.

We assume that a process of preparing Bins and planning Bins for the ports is active. This process may cause the following communication:

```
WMS: preparebin ( 15001, 15002, 15221, 15227, 15238, 15287, 16802 )
LPI:  BM,15002,T
LPI:  BM,15238,T
LPI:  BM,15001,T
WMS: appendportqueue( port_id=3, bin_id=15001, 15002, 15238 )
```

The process should continuously add new Bins to the "prepare" queue and the Port queues during operation.

A Port, (Port number 3) located at position (15,12), handling incoming Bins going out to an external belt system can cause the following communication:

```
WMS: openport( port_id=3 )
```

The Port is activated. The **AUTO** mode tells AutoStore to provide a stream of Bins to the Port, according to the Port's queue.

```
WMS: portcellallocate( port_id=3, x=15, y=12 )
```

The WMS requests a Robot to the Port.

```
WMS: portlocationready( port_id=3, x=15, y=12 )
```

The WMS confirms that position (15,12) is ready , and relinquishes control of the cell.

```
LPI:  BL,15001,1,15,12,1,P
```

AutoStore confirms via the Log Publisher Interface that Bin 15001 is put in the cell, and relinquishes control of the cell. Note that the depth should be ignored in the Port cell.

```
WMS: portcellallocate( port_id=3, x=15, y=12 )
```

The WMS requests a Robot to the Port.

```
WMS: portlocationready( port_id=3, x=15, y=12 )
```

The WMS confirms that position (15,12) is ready, and relinquishes control of the cell.

```
LPI:  BL,15002,1,15,12,1,P
```

AutoStore confirms via the Log Publisher Interface that Bin 15002 is put in the cell, and relinquishes control of the cell.

The process is repeated as long as the Port is active.

```
WMS: closeport( port_id=3 )
```

When the Port is no longer in use, the WMS should use closeport to release the resources that AutoStore plans for open Ports.

9.3.3 Port with Incoming Bins from Conveyor Belt

A Port (Port number 4) located at position (15,16), handling incoming Bins from an external belt system can be involved in the following communication:

```
WMS: openport( port_id=4, mode=INSERT )
```

The Port is activated. The mode parameter tells AutoStore that the Port is used to insert bins.

```
WMS: portcellallocate( port_id=4, x=15, y=16, getbin=1 )
```

The WMS requests an empty Robot to the Port.

```
WMS: portbinready( port_id=4, x=15, y=16, bin_id=12345 )
```

The WMS confirms that Bin 12345 is ready at position (15 ,16), and relinquishes control of the cell.

```
LPI:  BL,12345,1
```

AutoStore confirms via the Log Publisher Interface that the Bin is fetched by a Robot, and relinquishes control of the cell.

```
WMS: portcellallocate( port_id=4, x=15, y=16, getbin=1 )
```

The WMS requests an empty Robot to the Port.

```
WMS: portbinready( port_id=4, x=15, y=16, bin_id=22456 )
```

The WMS confirms that Bin 22456 is ready at the position (15,16), and relinquishes control of the cell.

```
LPI:  BL,22456,1
```

AutoStore confirms via the Log Publisher Interface that the Bin is fetched by a Robot, and relinquishes control of the cell.

The process is repeated as long as the Port is active.

```
WMS: closeport( port_id=4 )
```

When the Port is no longer in use, the WMS should use closeport to release the resources that AutoStore plans for the open Ports.

9.3.4 Port Exchange Bins to and from Grid

This example illustrates Bin exchange in a conveyor belt Port where the bin is delivered to the middle position.

We assume that a process of preparing Bins and planning Bins for the Ports is active. Alternatively tasks can be added via Task Interface.

This process may cause in the following communication:

```
WMS: create_taskgroup( taskgroup_id=1, ... , task_id=11, bin_id=1218,
task_id=12, bin_id=1200, task_id=13, bin_id=1136 )
```

The process should continuously add new tasks during operation.

Alternatively, the process could continuously add new Bins to the preparation queue and the Port queues during operation.

9.3.4.1 Normal Process

We start with an empty Port:

Port number 1 located at position (31,6) can cause the following communication:

```
WMS: openport( port_id=1 )
```

The Port is activated. Default mode is **OPEN**.

```
WMS: portcellallocate( port_id=1, x=31, y=6 )
```

The WMS requests a Robot to the Port.

```
WMS: openbin( port_id=1 )
WMS: portlocationready( port_id=1, x=31, y=6 )
```

The WMS confirms that position (31,6) is ready and relinquishes control of the cell.

```
LPI:  BL,1218,1,31,6,1,P
```

AutoStore confirms via the Log Publisher Interface that Bin 1218 is in Port.

Physical: *Port moves bin Middle -> Open*

The belt moves Bin 1218 from Middle to Open.

```
WMS/PI: portbinmode( port_id=1, bin_id=1218, bin_mode=OPEN )
LPI: BM,1218,O
```

The Port declares that Bin 1218 is **OPEN**, and AutoStore forwards the confirmation via the Log Publisher Interface.

```
WMS: portcellallocate( port_id=1, x=31, y=6 )
```

The WMS requests the next Robot to the Port.

Then the bin is exchanged with a new bin:

```
WMS: closebin( port_id=1, bin_id=1218, task_id=11, taskcomplete=1 )
```

Physical: *Port moves Bin Open -> Closed*

The belt moves Bin 1218 from Open to Closed.

```
WMS/PI: portbinmode( port_id=1, bin_id=1218, bin_mode=CLOSED,
content=1 )
LPI:  BM,1218,C
```

The Port declares that Bin 1218 is **CLOSED**, and AutoStore forwards the confirmation via the Log Publisher Interface.

```
WMS: openbin( port_id=1 )
WMS: portlocationready( port_id=1, x=31, y=6, getbin=1 )
```

The WMS confirms that the position (31,6) is ready , and relinquishes control of the cell.

```
LPI: BL,1136,1,31,6,1,P
```

AutoStore confirms via the Log Publisher Interface that Bin 1136 is in Port.

Physical: *Port moves next Bin Middle -> Open, and previous Bin Closed -> Middle*

The belt moves Bin 1136 from Middle to Open and Bin 1218 from Closed to Middle.

```
WMS/PI: portbinmode( port_id=1, bin_id=1136, bin_mode=OPEN )
LPI:  BM,1136,O
```

The Port declares that Bin 1136 is **OPEN**, and AutoStore forwards the confirmation via the Log Publisher Interface.

```
WMS/PI: portbinready( port_id=1, x=31, y=6, bin_id=1218 )
```

The WMS confirms that the previous Bin is ready to be fetched by Robot.

```
LPI:  BL,1218,1,29,21,471,G
```

AutoStore confirms via Log Publisher Interface that Bin 1218 is moved to Robot or a position different than the Port's position.

```
WMS: portcellallocate( port_id=1, x=31, y=6 )
```

The WMS requests a Robot to the Port.

Then the Bin is exchanged with a new Bin:

```
WMS: closebin( port_id=1, bin_id=1136, task_id=13, taskcomplete=1 )
```

Physical: *Port moves bin Open -> Closed*

The belt moves Bin 1136 from Open to Closed.

```
WMS/PI: portbinmode( port_id=1, bin_id=1136, bin_mode=CLOSED,
content=1 )
LPI: BM,1136,C
```

The Port declares that Bin 1136 is **CLOSED**, and AutoStore forwards the confirmation via the Log Publisher Interface.

```
WMS: openbin( port_id=1 )
WMS: portlocationready( port_id=1, x=31, y=6, getbin=1 )
```

The WMS confirms that the position (31 ,6) is ready , and relinquishes control of the cell.

```
LPI: BL,1200,1,31,6,1,P
```

AutoStore confirms via the Log Publisher Interface that Bin 1200 is in Port.

Physical: *Port moves next Bin Middle -> Open, and previous Bin Closed -> Middle*

The belt moves Bin 1200 from Middle to Open and Bin 1136 from Closed to Middle.

```
WMS/PI: portbinmode( port_id=1, bin_id=1200, bin_mode=OPEN ) LPI:
BM,1200,O
```

AutoStore confirms via the Log Publisher Interface that Bin 1200 is **OPEN**.

```
WMS/PI: portbinready( port_id=1, x=31, y=6, bin_id=1136 )
```

The WMS confirms that the previous Bin ready to be fetched by Robot.

```
LPI: BL,1136,1,31,1,141,G
```

AutoStore confirms via Log Publisher Interface that Bin 1136 is moved to Robot or a position different than the Port's position.

```
WMS: portcellallocate( port_id=1, x=31, y=6 )
```

The WMS requests a Robot to the Port.

The process of exchanging Bins is repeated.

The last Bin in the process is closed.

```
WMS: closebin( port_id=1, bin_id=1200, task_id=12, taskcomplete=1 )
```

Physical: *Port moves bin Open -> Closed*

The belt moves Bin 1200 from Open to Closed.

```
WMS/PI: portbinmode( port_id=1, bin_id=1200, bin_mode=CLOSED,
content=1 )
LPI: BM,1200,C
```

AutoStore confirms via the Log Publisher Interface that Bin 1200 is **CLOSED**.

```
WMS: closeport( port_id=1 )
```

When the Port is no longer in use, the WMS should use closeport to release the resources that AutoStore plans for the open Ports.

9.3.4.2 Repeated Bin

If the next openbin is the same as previous closebin, the Port will move the closed Bin instead of relinquishing control of the Port cell.

```
WMS: openbin( port_id=1 )
```

Physical: *Port moves Bin Closed -> Open*

The belt moves Bin 1200 from Closed to Open.

```
WMS/PI: portbinmode( port_id=1, bin_id=1200, bin_mode=OPEN )
LPI: BM,1200,O
```

AutoStore confirms via the Log Publisher Interface that Bin 1200 is OPEN.

```
WMS: closebin( port_id=1, bin_id=1200, task_id=21, taskcomplete=1 )
```

Physical: *Port moves Bin Open -> Closed*

The belt moves bin 1200 from Open to Closed.

```
WMS/PI: portbinmode( port_id=1, bin_id=1200, bin_mode=CLOSED,
content=1 ) LPI: BM,1200,C
```

AutoStore confirms via the Log Publisher Interface that Bin 1200 is CLOSED.

9.3.4.3 Starting with Bin in Port

We start the process with a port that has a Bin. We assume that Bin 1200 is in Port number 1.

```
WMS: openport( port_id=1, mode=OPEN )
```

The Port is activated. Default mode is **OPEN**.

```
WMS: portcellallocate( port_id=1, x=31, y=6 )
```

The WMS requests a Robot to the Port.

```
WMS: openbin( port_id=1 )
WMS: portlocationready( port_id=1, x=31, y=6, getbin=1 )
```

The WMS confirms that the position (31,6) is ready, and relinquishes control of the cell.

```
LPI:  BL,1152,1,31,6,1,P
```

AutoStore confirms via the Log Publisher Interface that bin 1152 is in Port.

Physical: *Port moves Bin Middle -> Open, and the Bin that was already in the Port Closed -> Middle*

The belt moves Bin 1152 from Middle to Open and bin 1200 from Closed to Middle.

```
WMS/PI: portbinmode( port_id=1, bin_id=1152, bin_mode=OPEN ) LPI:
BM,1152,O
```

AutoStore confirms via the Log Publisher Interface that bin 1152 is **OPEN**.

```
WMS/PI: portbinready( port_id=1, x=31, y=6, bin_id=1200 )
```

The Port confirms that the previous bin is ready to be fetched by robot.

```
LPI: BL,1200,1,31,1,1,G
```

AutoStore confirms via Log Publisher Interface that Bin 1200 is moved to Robot or a position different than the Port's position.

```
WMS: portcellallocate( port_id=1, x=31, y=6 )
```

The WMS requests a Robot to the Port.

The process of exchanging Bins is repeated.

9.4 Error Handling

The WMS will be notified via the Log Publisher Interface when a Port operation fails which requires the Port to ensure that the location or Bin is available and properly aligned.

Instead of issuing a confirmation of the operation, the WMS will be notified by a different segment and AutoStore will relinquish control of the cell.

Not all Port failures will result in these notifications being issued. However, one will be issued should the Port should retry an operation (such for instance as realigning the Bin). Other failures can be handled by Super Users via the **CONSOLE** and the WMS can receive confirmation after the problem is solved.

9.4.1 Failure when Port Retrieves Bin from Grid

When the Port retrieves a Bin from the Grid, it waits for confirmation by a BL segment:

```
WMS: portcellallocate( port_id=3, x=15, y=12 )
WMS: portlocationready( port_id=3, x=15, y=12 )
LPI: BL,15001,1,15,12,1,P
```

If the operation fails, AutoStore will respond with a Port Location Failed segment instead of confirming the operation:

```
WMS: portcellallocate( port_id=3, x=15, y=12 )
WMS: portlocationready( port_id=3, x=15, y=12 )
LPI: LF,1,15,12
```

When AutoStore reports Port Location Failed, the WMS should check and ensure that the location is ok (no other Bins blocking the physical position), and retry the operation.

The following communication reflects one failure before a successful retrieval:

```
WMS: portcellallocate( port_id=3, x=15, y=12 )
WMS: portlocationready( port_id=3, x=15, y=12 )
LPI: LF,1,15,12
WMS: portlocationready( port_id=3, x=15, y=12 )
LPI: BL,15001,1,15,12,1,P
```

9.4.2 Failure when Port Deliver Bin To Grid

When the Port delivers a Bin to the Grid, it waits for confirmation by a BL segment:

```
WMS: portcellallocate( port_id=4, x=15, y=16, getbin=1 )
WMS: portbinready( port_id=4, x=15, y=16, bin_id=12345 )
LPI: BL,12345,1
```

If the operation fails, AutoStore will respond with a Bin mode **TOGRID** segment instead of confirming the operation:

```
WMS: portcellallocate( port_id=4, x=15, y=16, getbin=1 )
WMS: portbinready( port_id=4, x=15, y=16, bin_id=12345 )
LPI: BM,12345,R
```

When AutoStore reports this failure, the WMS should check and ensure that the Bin is available and properly aligned, and retry the operation.

The following communication reflects one failure before a successful delivery:

```
WMS: portcellallocate( port_id=4, x=15, y=16, getbin=1 )
WMS: portbinready( port_id=4, x=15, y=16, bin_id=12345 )
LPI: BM,12345,R
WMS: portbinready( port_id=4, x=15, y=16, bin_id=12345 )
LPI: BL,12345,1
```

9.5 Port Interface Methods

9.5.1 Port Cell Allocate

The **portcellallocate** message requests delivery of a Bin to, or retrieval of a bin from, a specific port cell. It announces the next cell where a Bin should be delivered to or fetched from. These messages define the sequence of the Port cells to where AutoStore should deliver or fetch bins from.

9.5.1.1 Parameters

port_id: (mandatory)

Specifies the Port.

x: (mandatory)

Specifies the X position of the announced cell.

y: (mandatory)

Specifies the Y position of the announced cell.

getbin: (optional)

Specifies if a Bin should be fetched from the cell. Default false.

9.5.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "portcellallocate" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element x { xsd:short { minInclusive = "1" } },
      element y { xsd:short { minInclusive = "1" } },
      element getbin { xsd:boolean } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>portcellallocate</name>
   <params>
      <port_id>1</port_id>
      <x>26</x>
      <y>18</y>
      <getbin>1</getbin>
   </params>
</methodcall>
```

9.5.1.3 Error Codes

Error Code	Description

1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1009	The specified Port is not open.
1048	System is not available.
1066	Mandatory parameter x is missing.
1067	Parameter x is not valid (invalid format or out of range).
1068	Mandatory parameter y is missing.
1069	Parameter y is not valid (invalid format or out of range).
1070	Mandatory parameter getbin is missing.
1071	Parameter getbin is not valid (invalid format or out of range).
1072	The specified position does not belong to the specified Port.
1074	Location is ready.

9.5.2 Port Location Ready

The **portlocationready** message declares that AutoStore can deliver a Bin to the specified position. The Port relinquishes control of the cell.

The message includes a flag that indicates whether or not AutoStore should retrieve another Bin from the cell immediately after the Bin has been delivered.

9.5.2.1 Parameters

port_id: (mandatory)

Specifies the Port.

x: (mandatory)

Specifies the X position of the ready location.

y: (mandatory)

Specifies the Y position of the ready location.

getbin: (optional)

Specifies if another Bin should be fetched from the cell after the Bin has been put. Default false.

9.5.2.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "portlocationready" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element x { xsd:short { minInclusive = "1" } },
      element y { xsd:short { minInclusive = "1" } },
```

```
            element getbin { xsd:boolean } } ?
      }
   }
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>portlocationready</name>
   <params>
      <port_id>1</port_id>
      <x>26</x>
      <y>18</y>
      <getbin>1</getbin>
   </params>
</methodcall>
```

9.5.2.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1009	The specified Port is not open.
1048	System is not available.
1066	Mandatory parameter x is missing.
1067	Parameter x is not valid (invalid format or out of range).
1068	Mandatory parameter y is missing.
1069	Parameter y is not valid (invalid format or out of range).
1070	Mandatory parameter getbin is missing.
1071	Parameter getbin is not valid (invalid format or out of range).
1072	The specified position does not belong to the specified Port.

9.5.3 Port Bin Ready

The **portbinready** message confirms that the Bin is ready in the cell, and can be fetched by AutoStore.

The Port relinquishes control of the cell.

9.5.3.1 Parameters

port_id: (mandatory)

Specifies the Port.

x: (mandatory)

Specifies the X position of the cell.

y: (mandatory)

Specifies the Y position of the cell.

bin_id: (mandatory)

Identifies the Bin that is ready in the cell.

content: (optional, but mandatory when bin_type is set)

Sets the Bin's content. If this parameter is omitted, the bin will keep its previous content.

bin_type: (optional, but mandatory if the bin is not defined yet)

Specifies the Bin's type.

This parameter is mandatory if the Bin is not defined (the first time the Bin is inserted). In installations where Bins are removed and inserted, the bin_type parameter is not required in subsequent inserts.

9.5.3.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "portbinready" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element x { xsd:short { minInclusive = "1" } },
      element y { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } },
      element content { xsd:short { minInclusive = "1" } } ?,
      element bin_type { xsd:short { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>portbinready</name>
   <params>
      <port_id>1</port_id>
      <x>26</x>
      <y>18</y>
      <bin_id>25738</bin_id>
   </params>
</methodcall>
```

9.5.3.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).

1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1024	Mandatory parameter bin_id is missing.
1029	The specified Bin is already inserted.
1048	System is not available.
1058	Bin is still outside.
1066	Mandatory parameter x is missing.
1067	Parameter x is not valid (invalid format or out of range).
1068	Mandatory parameter y is missing.
1069	Parameter y is not valid (invalid format or out of range).
1072	The specified position does not belong to the specified Port.

9.5.4 Port Bin mode

The **portbinmode** message sets the Bin mode to either **OPEN**, **CLOSED,** or **OUTSIDE**.

9.5.4.1 Parameters

port_id: (mandatory)

Identifies the Port.

bin_id: (mandatory)

Identifies the Bin.

bin_mode: (mandatory)

Sets the Bin mode. Can be **OPEN, CLOSED, or OUTSIDE**.

Setting the Bin mode to **OUTSIDE** is only allowed if the current Bin mode is **TOGRID**.

content: (optional)

Sets the Bin content.

If this parameter is not provided then the Bin's content is not changed.

9.5.4.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "portbinmode" },
   element params {
      element port_id { xsd:short { minInclusive = "1" } },
      element bin_id { xsd:int { minInclusive = "1" } },
      element bin_mode { "OPEN" | "CLOSED" | "OUTSIDE" },
      element content { xsd:short { minInclusive = "1" } } ?
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>portbinmode</name>
   <params>
      <port_id>1</port_id>
      <bin_id>22344</bin_id>
      <bin_mode>CLOSED</bin_mode>
      <content>1</content>
   </params>
</methodcall>
```

9.5.4.3 Error Codes

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1024	Mandatory parameter bin_id is missing.
1048	System is not available.
1075	Parameter bin_mode is not valid (invalid format or out of range).

10. Connecting Grids with Bin Lift

AutoStore supports connecting two grids together with Bin Lifts, and exchange bins between the grids.

This chapter describes the integration of two systems that are connected with Bin Lifts.

Both Task Interface and Bin Interface support this functionality.

10.1 Introduction

The two Grids are two separate AutoStore systems with their own Control System and application servers (API endpoints). The WMS must integrate with both systems separately.

In the two separate grids, the transfer positions that are connected with a Bin Lift are handled like Ports, and two Ports connected with a Bin Lift must have the same port_id. Opening, inserting and removing bins to/from grids is done automatically by the system, but the WMS must define jobs (taskgroups or port queues) for these G2G transfer ports.

It is WMS responsibility to ensure that the balance of bins between the grids is maintained correctly.

The WMS must use Log Publisher in both systems to be able to be able to confirm and complete tasks fulfilled.

10.2 Exchanging Bins between the Grids

To activate the bin exchange between two grids, both ports must be opened in mode SWAP.

Using Bin Interface: Both ports must have a Port Queue that specifies which bins to move to the other grid.

Using Task Interface: Both ports must be opened with categories having task groups that specify which bins to move to the other grid.

Using port mode SWAP will ensure that the bin balance between grids are kept stable. Please note that at a complete restart, the balance could become 1 off, so the WMS should monitor the bin balance between the grids. To make the process run, both ports must be activated and both ports must have a work queue (taskgroups or port queue). If one of the ports runs out of work, then the exchange process will stop and wait.

10.3 Transfer Bins to the other Grid

To transfer bins from one grid to the other grid (stream of bins in one direction), the port in the source grid must be opened in mode OPEN, and the port in the destination grid must be opened in mode INSERT.

Using Bin Interface: The port in the source grid must have a port queue that specifies which bins to move to the other grid.

Using Task Interface: The port in the source grid must be opened with categories having task groups that specify which bins to move to the other grid.

To make the process run, both ports must be activated and the port in the source grid must have a work queue (taskgroups or port queue).

11. Miscellaneous Interface Methods

This chapter describes miscellaneous interface methods that are normally not required by the WMS.

These methods are available for both the Bin Interface and the Task Interface.

11.1 Bin Definition

11.1.1 Create Bin

The **createbin** method defines a new Bin. The new Bin is defined with Bin mode OUTSIDE.

The specified Bin must not already exist.

Normally, new Bins are defined when the Bins are first inserted by using the insertbin method (or portbinready for custom Ports).

In rare cases (typically during development), it is desirable that the Bin is defined before it is first inserted. It is not possible to use an undefined Bin in a Port queue or task group. If the bin is added to a Port queue before it is inserted it is more likely that the Bin is transferred directly to that Port instead of first being put temporarily in the Grid.

11.1.1.1 Parameters

bin_id: (mandatory)

Identifies the Bin that should be defined.

bin_type: (mandatory)

Specifies the Bin type of the new Bin.

11.1.1.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "createbin" },
   element params {
      element bin_id { xsd:int { minInclusive = "1" } },
      element bin_type { xsd:short { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>createbin</name>
```

```
   <params>
      <bin_id>14861</bin_id>
      <bin_type>2</bin_type>
   </params>
</methodcall>
```

11.1.1.3 Error Code

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1024	Mandatory parameter bin_id is missing.
1029	The specified Bin is already inserted.
1073	Parameter bin_type is not valid (invalid format or out of range).
1086	Mandatory parameter bin_type is missing.

11.1.2 Delete Bins

The **deletebin** method deletes the definition of a Bin.

The specified Bin must have Bin mode **OUTSIDE**.

11.1.2.1 Parameters

bin_id: (mandatory)

Identifies the Bin that should be deleted.

11.1.2.2 Message Format

Compact syntax:

```
element methodcall {
   element name { "deletebin" },
   element params {
      element bin_id { xsd:int { minInclusive = "1" } }
   }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>deletebin</name>
   <params>
      <bin_id>14861</bin_id>
   </params>
</methodcall>
```

11.1.2.3 Error Codes

Error Code	Description
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1024	Mandatory parameter bin_id is missing.
1076	Bin is not outside.

11.2 System Configuration

11.2.1 Force Charge

The **forcecharge** method sets the percentage of robots that should be forced to charge batteries.

The only allowed percentages are 0% (default), 50% and 100%.

This configuration only applies to the R5 robots. If the installation has a mix of robot types, the number of R5 robots is calculated as 100%.

11.2.1.1 Parameters

charge_percent: (optional)

If the parameter charge_percent is provided, the percentage of Robots that should be forced to charge is set to the specified percentage.

If the parameter charge_percent is not provided, the percentage of Robots that should be forced to charge is set to the default value (0%).

11.2.1.2 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "forcecharge" },
   element params {
      element charge_percent { xsd:int { minInclusive = "0" } } ?,
   }
}
```

Compact syntax (Response):

```
element response {
   element params { }
}
```

Request example:

```
<?xml version="1.0"?>
<methodcall>
   <name>forcecharge</name>
   <params>
      <charge_percent>50</charge_percent>
```

```
   </params>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params />
</response>
```

11.2.1.3 Error Codes

Error Code	Description
1088	Parameter charge_percent is not valid (invalid format or out of range).

11.2.2 Set Configuration

The **setconfiguration** method updates the configurations of the Planner module.

The following settings can be updated:

Configuration Name	Configuration Value
preparationspan	Integer value that specifies the maximum amount of Bins prepared, calculated as a percentage of the defined grid storage cells.
prepareemptybin	Integer value that specifies the amount of empty bins (content=0) prepared, calculated as a percentage of the defined grid storage cells.
emptybincontent	Integer value that specifies the content code used for specifying an empty bin. Multiple content codes can be specified by separating these with a comma (,).
emptybinfactor	String value used to override the default emptybin distribution. The value consists of the content code and the factor for this content code. These values are separated by a comma (,).

11.2.2.1 Parameters

configname: (mandatory)

The name of the configuration that should be updated.

configvalue: (mandatory)

The new value of the specified configuration.

11.2.2.2 Message Format

Compact syntax:

```
element methodcall {
```

```
      element name { "setconfiguration" },
      element params {
         element configname { xsd:string },
         element configvalue { xsd:string }
      }
   }
```

Request example (preparationspan configuration):

```
<?xml version="1.0"?>
<methodcall>
   <name>setconfiguration</name>
   <params>
      <configname>preparationspan</configname>
      <configvalue>10</configvalue>
   </params>
</methodcall>
```

Request example (prepareemptybin configuration):

```
<?xml version="1.0"?>
<methodcall>
   <name>configureport</name>
   <params>
      <configname>prepareemptybin</configname>
      <configvalue>10</configvalue>
   </params>
</methodcall>
```

Request example (emptybincontent configuration):

```
<?xml version="1.0"?>
<methodcall>
   <name>configureport</name>
   <params>
      <configname>emptybincontent</configname>
      <configvalue>99</configvalue>
   </params>
</methodcall>
```

Request example (emptybinfactor configuration):

```
<?xml version="1.0"?>
<methodcall>
   <name>configureport</name>
   <params>
      <configname>emptybinfactor</configname>
      <configvalue>99,10</configvalue>
   </params>
</methodcall>
```

11.2.2.3 Error Codes

Error Code	Description
1084	Invalid parameter configname.
1085	Invalid parameter configvalue.

11.3 Query Methods

11.3.1 Get System State

The **getsystemstate** method retrieves information about the current mode of the system, and the currently set forcecharge setting.

11.3.1.1 Parameters

No parameters are required for this method.

11.3.1.2 Response

mode:

The current mode of the system.

Mode	Description
20	Running
30	Service
35	Alert
70	Stopped

stopcode:

A stop code that could provide information about why the system stopped.

chargepercent:

The currently set percentage of robots that should be forced to charge batteries.

11.3.1.3 Message Format

Compact syntax (Request):

```
element methodcall {
   element name { "getsystemstate" }
}
```

Compact syntax (Response):

```
element response {
   element params {
      element mode { xsd:int },
      element stopcode { xsd:int },
      element chargepercent { xsd:int }
   }
}
```

Request example:

```
<?xml version="1.0"?>
```

```
<methodcall>
   <name>getsystemstate</name>
</methodcall>
```

Response example:

```
<?xml version="1.0"?>
<response>
   <params>
      <mode>20</mode>
      <stopcode>0</stopcode>
      <chargepercent>50</chargepercent>
   </params>
</response>
```

12. Log Publisher Interface

Every event in the AutoStore system is reflected in the AutoStore Log.

The Log Publisher Interface provides an event stream that is a selected subset of AutoStore Log. External modules such as the WMS may listen to this event stream.

12.1 Technical Interface

The Log Publisher Interface provides a TCP/IP server that sends the event stream to its clients.

A client that connects to this server on a specified port will receive a stream of data that reflects the AutoStore Log.

This data stream contains readable characters, so it is possible to connect with a standard telnet client in order to view it.

Each event (event segment) consists of:

- a tag that identifies an event
- optional parameters separated by commas and terminated by a **newline** (CRLF)

12.2 Initial Download

When a client connects to the system, the Log Publisher Interface will send a set of event segments that reflects the current state of the AutoStore system. This enables the client to have a complete copy of the current state.

12.3 Event Segments

Overview of Supported segments

Event Segment	Tag
Download Start	DS
Download Complete	DC
Time	T
Time Factor	F
Bin Location	BL
Bin Mode	BM
Port Location Failed	LF
Port Mode	PM
Cell Lock	CL
Lift Cell Lock	LCL
Robot Error	RE

Robot Charging	RC
Robot Unavailable	R
Port Mode	PM
Port Error	PE
Port Definition	PD
Port Task Groups	PTG
Pick To Light	PTL
Drawer State	PDS
System Mode	SM
Alert	AL

12.3.1 Download Start

The **Download Start** segment indicates that a new initial download has started.

```
DS
```

12.3.2 Download Complete

The **Download Complete** segment indicates that the initial download has completed (segments for all Bins are sent).

```
DC
```

12.3.3 Time

The **Time** segment is published every second. The **Time** segment that is published every minute will also include the current unixtime.

Time factor speed-up during simulation will be reflected by this segment (will appear more often).

```
T               // one second increment
T,unixtime      // sets current time
```

12.3.4 Time Factor

The **Time Factor** segments reflects the speed of simulation. (100 = normal).

```
F,timefactor
```

The Time Factor segment is always included in the initial download.

12.3.5 Bin Location

The **Bin Location** segment reflects changes in the physical location of the Bin.

When the Bin is on a Robot, then the position (`xpos,ypos`) is omitted or zero (0,0).

```
BL,bin_id,grid_id          // bin on robot
```

```
BL,bin_id,grid_id,xpos,ypos,depth   // new bin location
BL,bin_id,grid_id,xpos,ypos,depth,mode
BL,bin_id,grid_id,xpos,ypos,depth,mode,content,height
```

The depth for a given Bin is measured in mm from the top of the Grid framework to the top of the Bin. Specifically, the top of the Grid track is at height 0. If a Bin is temporarily placed on top of the Grid, for instance while Bins are being reshuffled during a digging operation, then the Bin has a negative depth value.

The Bins at the same level will have approximately the same depth, but this will vary due to the bins and the floor. The bins at the upper normal level (not on top of the Grid) will normally have a depth between 105 mm and 145 mm.

In a Port cell, the depth parameter will not reflect the actual depth and should be ignored.

The height corresponds with the bin_type.

12.3.6 Bin Mode

The **Bin Mode** segment reflects changes in the Bin Mode only.

```
BM,bin_id,mode
```

Bin Mode PREPARED is normally only used for Bin Interface, but it is possible to activate this mode for Task Interface in order to monitor the preparation process.

12.3.7 Port Location Failed

The **Port Location Failed** segment indicates that AutoStore was unable to deliver a Bin to the specified location.

```
LF,grid_id,xpos,ypos
```

12.3.8 Port Mode

The **Port Mode** segment reflects changes in the port mode.

```
PM,port_id,mode
```

12.3.9 Port Definition

The **Port Definition** segment contains the location and type of the ports.

```
PD,port_id,port_type,xpos,ypos           // Port with one cell
PD,port_id,port_type,xpos,ypos,xpos,ypos    // Port with two cells
PD,port_id,port_type,xpos,ypos,...,xpos,ypos   // Port with multiple
cells
```

12.3.10 Cell Lock

The **Cell Lock** segment reflects changes in the accessibility of grid cells. If a robot displays an error, all cells that might be occupied by this robot are locked for other robots. It is also possible to lock specific cells manually. These cells are reported via the Cell Lock segment. Once these cells are available again, another Cell Lock segment will be sent out with a "clear" flag.

```
CL,1,L,xpos,ypos // Cell temporarily locked by Delayed Stop
CL,1,P,xpos,ypos // Cell manually locked by user
CL,1,C,xpos,ypos // Clear cell lock
```

Cells are manually locked by the Cell Lock wizard in the Console. Cell mode "Blocked" (Grid Definition) is not reflected by the CL segment.

The cell lock state for unlocked cells is not transmitted on the initial download. Therefore all cell locks must be cleared when receiving a DS segment. The cells which are still locked will be transmitted during the initial download.

12.3.11 Lift Cell Lock

The **Lift Cell Lock** segment reflects changes in the lifting accessibility of grid cells. When a lifting operation fails, for instance because of an overfilled bin, the system can suspend the cell for lifting until the problem is resolved.

```
LCL,xpos,ypos,BOTH  // Cell is locked for lifting operations
LCL,xpos,ypos,CLEAR // Cell is no longer locked for lifting operations
```

The lift cell lock state for unlocked cells is not transmitted on the initial download. Therefore all lift cell locks must be cleared when receiving a DS segment. The cells which are still locked for lifting will be transmitted during the initial download.

12.3.12 Port Error

The **Port Error** segment reflects the error status of a port.

```
PE,port_id,1 // Port has an error
PE,port_id,0 // Error solved; Port has no error
```

12.3.13 Robot Charging

The **Robot Charging** segment reflects the charge status of a robot.

```
RC,robot_id,1 // Robot charging
RC,robot_id,0 // Robot not charging
```

12.3.14 Robot Error

The **Robot Error** segment reflects the error status of a robot.

```
RE,robot_id,1 // Robot has an error
RE,robot_id,0 // Error solved; Robot has no error
```

12.3.15 Robot Unavailable

The **Robot Unavailable** segment reflects the status of a robot. A robot is designated as unavailable if the robotmode is SERVICE, PARKED, or CONSOLE

```
R,robot_id,1   // Robot unavailable
R,robot_id,0   // Robot available
```

12.3.16 Port Task Groups

The **Port Task Groups** segments list the next Task Groups of the corresponding port.

```
PTG,port_id,taskgroup_1,taskgroup_2,.........,taskgroup_x
```

12.3.17 Pick To Light

The **Pick To Light** segments reflects changes to the Pick-To-Light visualization requests.

```
PTL,source_port_id,target_port_id,source_bin_id,target_bin_id,
source_compartment,target_compartment,quantity,status
PTL,source_port_id,target_port_id   // delete PTL
```

The status is one of: 'R' (Registered), 'A' (Active), 'C' (Confirmed).

Implicit deletion of PTL caused by the methods *closebin* and *closeport* is not reflected by Log Publisher.

12.3.18 Drawer State

The Drawer State segments reflect changes in the Port's physical drawer state.

```
PDS,port_id,drawer_position,snap_lock
PDS,1,O,E     // port 1, drawer open, lock enabled
PDS,1,C,D     // port 1, drawer closed, lock disabled
PDS,1,U,U     // port 1, unknown position and lock state
```

12.3.19 System Mode

The **System Mode** segment reflects changes in the System Mode.

```
SM,systemmode
```

12.3.20 Alert

The Alert segment is sent if the connection to the AutoStore system or Backup module is lost (or delayed). When the connection to AutoStore is re-established the current system mode is published.

If the connection between AutoStore Log Publisher and AutoStore Driver is lost for more than 20 seconds, the event code **NOC** is published.

If the connection between AutoStore Driver and AutoStore Backup is either too slow or lost for more than 60 seconds, the event code **BCK** is published.

In some error situations, the AutoStore XHandler may postpone stopping the system. The event code **DSTP** is published in this case.

```
AL,code
```

12.4 Value Codes

Most of the parameters that belong to the event segments are integer values.

Some parameters that have a string representation in the XML interface are encoded as single characters in the Log Publisher Interface in order to have a compact format.

Bin Mode:

Mode	Code in Log Publisher Interface
GRID	G
PREPARED	T

PORT	P
TOGRID	R
OPEN	O
CLOSED	C
OUTSIDE	X
FORECAST	F

Port Mode:

Mode	Code in Log Publisher Interface
OPEN	O
INSERT	I
AUTO	A
CLOSED	C
SEQUENCE	S
DISABLED	D
MULTIPORT_PUPPET	M

Port Type:

Type	Code in Log Publisher Interface
Conveyor Port	CO
Carousel Port	CA
Swing Port	SW
Relay Port	RE
Transfer Cell / Custom Port	TC
Fusion Port	FU
MultiPort	MU
PickUpPort	PI

System Mode:

System mode	Code in Log Publisher Interface
RUNNING	RUN
SERVICE	SVC
ALERT	ALT
STOPPING	DSTP
STOPPED	STP

Alert:

Code	Code in Log Publisher Interface

NO_CONNECTION	NOC
BACKUP_SEQ	BCK
DELAYED_STOP	DSTP

13. AutoStore Internals and Configurations

This chapter contains miscellaneous topics about AutoStore functionality and configurations that WMS developers should be aware of.

13.1 Port Carousel Behavior

Due to the way the carousel arms work together, repeating the same Bin with only one other Bin in between is very inefficient.

It is strongly recommended to avoid the following Bin sequence on Port Carousels:

```
... Bin A - Bin B - Bin A ...
```

> **NOTICE**
> **Even though the above sequence will work, and may sometimes be required by the WMS, it should be avoided whenever possible in order to obtain optimal performance.**

13.2 Port Configurations

Robot Queue

This configuration specifies how many Bins that will be queued towards each Port. (This configuration was previously called Port Queue.)

For Planner, the default value is 2. For Router, the default is Automatic. Router will automatically adjust the muber of robots allocated to a port, but that can be overruled by this configuration.

The value has an upper limit that is equal to the number of Port cells and buffer cells at the Port.

Value	Description
0	This would cause no Robots to stand and wait. So when a new Bin is requested a Robot would pick it up and then deliver it. This will cause long delivery times and is only appropriate for very slow inspection or putaway Ports.
1	This allows one Robot to wait above the Port with the next bin. This is appropriate for most putaway operations which usually take longer on the user side.
2	This allows one Robot to wait above the Port and the second above the buffer position. This is the current default, and appropriate for medium performance Ports.
3 or more	This is for fast-picking Ports. (The value is often about the same as the number of picks per minute, but other considerations may also apply.)

Minimum Cycle Time

Theoretical minimum time a Bin spends in a Port (in seconds). Default value is 8.

13.3 Behavior when few available Tasks or Bins

AutoStore is optimized for a high throughput environment that has sufficient work for the open ports.

In situations where there are very few task groups in the system, or very few bins that corresponds with the requested content codes, AutoStore may assign the work to one of the ports, instead of balancing them on multiple open ports. In these situations some ports are not receiving bins, because the few bins or task groups are already internally assigned to another port.

This behavior can typically be observed during simple tests with the WMS SDK.

14. Appendix

14.1 Error Code Overview

Error Code	Description
1000	The specified Port is not available.
1001	The specified Port is closing.
1002	The specified Port is already open.
1003	Mandatory parameter port_id is missing.
1004	Parameter port_id is not valid (invalid format or out of range).
1006	Parameter content is not valid (invalid format or out of range).
1007	Parameter category is not valid (invalid format or out of range).
1008	Parameter req_time is not valid (invalid format or out of range).
1009	The specified Port is not open.
1010	Parameter bin_id is not valid (invalid format or out of range).
1011	The specified Bin does not exist.
1012	Parameter task_id is not valid (invalid format or out of range).
1013	Parameter taskgroup_id is not valid (invalid format or out of range).
1014	Parameter shipment is not valid (invalid format or out of range).
1015	There are no tasks or Bins that fulfill the Port select requirements.
1016	There are no Bins ready that fulfill the Port select requirements.
1017	The specified task does not exist.
1018	The specified task group does not exist.
1019	The specified task group is selected by another Port.
1020	The openbin conflicts with a previous openbin call that is not yet closed.
1021	The openbin conflicts with the Port's selected shipment.
1022	The openbin conflicts with the Port's selected task group.
1023	Multiple selection parameters are not allowed.
1024	Mandatory parameter bin_id is missing.
1025	Parameter taskcomplete is not valid (invalid format or out of range).
1026	The Port does not have a selected Bin.
1027	The Bin is not yet ready in the Port.
1028	The Bin and task parameters conflict with the open Bin and task.
1029	The specified Bin is already inserted.

1030	The Port has a selected Bin.
1031	Mandatory parameter taskgroup_id is missing.
1032	The task group already exists.
1033	The task already exists.
1034	Mandatory parameter category is missing.
1035	Mandatory parameter req_time is missing.
1036	Mandatory parameter priority is missing.
1037	Parameter priority is not valid (invalid format or out of range).
1038	Parameter start_time is not valid (invalid format or out of range).
1039	Parameter suspended is not valid (invalid format or out of range).
1040	Mandatory parameter task_id is missing.
1041	Parameter item is not valid (invalid format or out of range).
1042	Parameter order is not valid (invalid format or out of range).
1043	The task group has no tasks.
1044	The task group is selected by a Port.
1045	The task is selected by a Port.
1046	XML parsing error.
1047	Unknown method.
1048	System is not available.
1049	Response overflow.
1050	Too many categories or content codes.
1051	Conflicting category and content.
1052	Parameter suspendtaskgroup is not valid (invalid format or out of range).
1053	Parameter mintaskgroupid is not valid (invalid format or out of range).
1054	Parameter maxtaskgroupid is not valid (invalid format or out of range).
1055	Parameter min_bin_id is not valid (invalid format or out of range).
1056	Parameter max_bin_id is not valid (invalid format or out of range).
1057	Parameter estpicktime is not valid (invalid format or out of range).
1058	Bin is still outside.
1059	Too many tasks included.
1060	Parameter port_mode is not valid (invalid format or out of range).
1061	Parameter target_grid is not valid (invalid format or out of range).
1062	Port queue is full.
	The maximum size of a port's port queue is 1000 bins.
1063	Bin preparation queue is full.
	The maximum size of the bin preparation queue is 20000 bins.
1064	Mandatory parameter element Bins are missing.

1065	Parameter all is not valid (invalid format or out of range).
1066	Mandatory parameter x is missing.
1067	Parameter x is not valid (invalid format or out of range).
1068	Mandatory parameter y is missing.
1069	Parameter y is not valid (invalid format or out of range).
1070	Mandatory parameter getbin is missing.
1071	Parameter getbin is not valid (invalid format or out of range).
1072	The specified position does not belong to the specified Port.
1073	Parameter bin_type is not valid (invalid format or out of range).
1074	Location is ready.
	The message *portlocationready* has been sent. Before sending the next *portcellallocate* for the same cell, the Bin must be confirmed in the Port (confirmation from Log Publisher).
1075	Parameter bin_mode is not valid (invalid format or out of range).
1076	Bin is not outside.
1077	Maximum number of Bins already inserted.
1078	Bin is in Port.
	The messages *forecastbin* and *cancelforecastbin* will be rejected if the Bin is in a Port.
1079	Illegal Bin mode.
	The message *forecastbin* will be rejected if the current bin mode is not GRID. The message *cancelforecastbin* will be rejected if the current bin mode is not FORECAST.
1080	Method is not supported.
	The method is not supported by some of the other modules. For example, the method may requires a newer version of AutoStore Driver or AutoStore Planner.
1081	Mandatory parameter index is missing.
1082	Parameter index not valid (invalid format or out of range).
1083	No room for given content.
1084	Invalid parameter configname.
1085	Invalid parameter configvalue.
1086	Mandatory parameter bin_type is missing.
1087	Illegal Port type.
1088	Parameter charge_percent is not valid (invalid format or out of range).
9997	Undefined error (internal server error).
9998	Unknown reject code.
9999	Driver timeout.
	The method may or may not have been completed. The WMS must handle both situations. The interface did not receive confirmation from the driver within the time limit (might be a network issue between application server and ASCS).

	In a simulation environment, this fault can be caused by a too high time factor or workload on the simulation server.

14.2 Property Overview

Name	Data Type	Description
bin_id	Integer between 1 and 1000000.	Identifies a bin.
category	Integer between 1 and 32767.	Category code.
content	Integer between 1 and 32767.	Content code.
item	Integer between 1 and 2147483647.	Task grouping within a task group.
order	Integer between 1 and 2147483647.	Task order within a task group.
port_id	Integer between 1 and 1024.	Identifies a port.
priority	Integer between 1 and 32767.	Task group priority.
req_time	Date and time. Between 1971.01.01 and 2038.01.18.	Requested time.
shipment	Integer between 1 and 2147483647.	Identifies a shipment.
start_time	Date and time. Between 1971.01.01 and 2038.01.18.	Earliest pick time.
suspended	Boolean.	Task group suspended.
task_id	Integer between 1 and 2147483647.	Identifies a task.
taskgroup_id	Integer between 1 and 2147483647.	Identifies a task group.

15. Index

A

About this manual 9
Add Task 43
Alert 144
Allocate Cell 117
Alternative flow 97
Append Port Sequence 58
Append to Port Queue 75
Appendix 149
Automatic Preparation of Empty Bins 20
AutoStore Internals and Configurations 147

B

Balancing Categories 25
Basic Concepts 60, 18, 23
Begin Group in Port Queue 77
Behavior when few available Tasks or Bins 148
Bin 18
Bin Content Code 111, 19
Bin Definition 133
Bin Interface 60, 13
Bin Location 141
Bin Mode 142
Bin Mode Forecast 106
Bin Preparation 60
Bin Preparation Methods 70
Bin Properties 18

C

Cancel Forecast Bin 108
Cancel Prepare Bin 72
Cancel Task 44
Cancel Taskgroup 42
Cell Content Code 111
Cell Lock 142
Changes to the Manual 10
Close Bin 64, 30
Close Port 65, 31
Comparison 14
Compartment Definition 95
Configurations 112, 20
Configure Port 99
Connecting Grids with Bin Lift 132
Contacting us 10
Content Code Example 19
Content Code Ranges 112
Content Consumers 113
Controlling Bin Locations 111
Controlling Bin time outside its Zone 115
Controlling the Sequence of Task Groups 55
Conventions and Standards 9
Copyright, Intellectual Property, and Disclaimer 4
Create Bin 133
Create Taskgroup 38

D

Defining the Zones 115
Delete Bins 134
Description 106
Disable MultiPort 98
Download Complete 141
Download Start 141
Drawer State 144

E

Enable MultiPort 98
End Group in Port Queue 77
Error Code 67, 70, 134
Error Code Overview 149
Error Codes 42, 44, 45, 46, 48, 50, 53, 55, 58, 59, 62, 64, 65, 66, 69, 71, 72, 73, 75, 76, 77, 78, 80, 81, 84, 85, 86, 90, 93, 95, 98, 99, 101, 102, 104, 105, 107, 109, 115, 126, 128, 129, 131, 135, 136, 137, 27, 29, 31, 32, 33, 35, 36, 37, 39, 41
Error Handling 125
Event Segments 140
Examples 119
Exchange Bins to and from Grid 117
Exchanging Bins between the Grids 132

F

Failure when Port Deliver Bin To Grid 125
Failure when Port Retrieves Bin from Grid 125

Feature Compatibility 16
Features that require AS Router 16
Flush Port Bin 104
Force Charge 135
Forecast Bin 106
Forecast Bin Preparation 106
Forecast Methods 106
Fusion Port 90

G

General Precautions 11
General Safety Information 10
Get Bin Information 45, 78
Get Bin Location 47, 80
Get Bin State 49, 82
Get Drawer State 89
Get Next Task Groups 53
Get Port Configuration 101
Get Port Queue 85
Get Port Status 36
Get Port Weight 102
Get Preparation Queue 84
Get System State 138
Get Task Information 51
Grid 22
Guidelines 109

H

Hide Pick-To-Light 94

I

Implementing Cut-Off Time 55
Imprint 3
Initial Download 140
Insert Bin 66, 32
Insert Prepare Bin 71
Interface 112
Interface Methods 114
Introduction 12, 91, 97, 111, 132

L

Lift Cell Lock 143
Log Publisher Interface 140

M

Main Configurations 112
Managing Frozen Zone 115
Message flow 96
Message Flow 118
Message Format 43, 44, 45, 47, 49, 52, 54, 57, 59, 62, 63, 65, 66, 67, 68, 70, 71, 72, 73, 74, 76, 77, 78, 79, 81, 83, 84, 85, 89, 92, 94, 98, 99, 100, 101, 103, 104, 107, 108, 114, 126, 127, 129, 130, 133, 134, 135, 136, 138, 26, 28, 30, 32, 33, 34, 35, 37, 38, 41, 42
Miscellaneous Interface Methods 133
Miscellaneous Port Methods 99
MultiPort configuration of the Fusion Port 97

N

Normal Process 122

O

Open Bin 62, 27
Open Port 61, 26
Overriding the next Task Group 55

P

Parameters 43, 44, 45, 47, 49, 51, 54, 57, 58, 61, 63, 64, 66, 67, 68, 69, 70, 71, 73, 74, 75, 77, 77, 78, 80, 82, 84, 85, 89, 92, 94, 98, 98, 100, 101, 102, 104, 106, 108, 114, 126, 127, 128, 130, 133, 134, 135, 136, 138, 26, 28, 30, 31, 32, 34, 35, 36, 38, 40, 42
Performance Guidelines 26
Pick To Light 144
Picking Process with Bin Interface 88
Picking Process with Task Interface 88
Pick-To-Light API 91
Pick-To-Light Example 95
PickUpPort 89
Port 20
Port Bin mode 130
Port Bin Ready 128
Port Carousel Behavior 147
Port Cell Allocate 126
Port Configurations 147
Port Definition 142
Port Deliver Bin to Grid 118
Port Error 143
Port Exchange Bin with Grid 119
Port Exchange Bins to and from Grid 121
Port Interface 117
Port Interface Methods 126
Port Location Failed 142
Port Location Ready 127
Port Methods 61, 26
Port Mode 21, 142
Port Mode SEQUENCE 56
Port Process 117
Port Processes 87

Port Properties 20, 24
Port Queue 60
Port Queue Group 61
Port Retrieve Bin from Grid 118
Port Task Groups 143
Port with Incoming Bins from Conveyor Belt 121
Port with Outgoing Bins to Conveyor Belt 119
Port with Outgoing Bins to Conveyor Belt (mode AUTO) 120
Prepare Bin 70
Properties 24
Property Overview 152
Putaway Process 87

Q

Query Methods 45, 78, 138

R

Read this first 9
Remove Bin 68, 34
Repeated Bin 124
Request 15
Response 45, 47, 49, 51, 54, 63, 79, 80, 82, 15, 84, 85, 89, 101, 103, 138, 28, 36
Robot Charging 143
Robot Error 143
Robot Unavailable 143

S

Select Requirements 24
Selected Bin 25
Set Configuration 136
Set Port Queue 74
Set Port Sequence 57
Show Pick-To-Light 92
Signs and Notices 10
Software Release April 2023 17
Software Release October 2022 17
Software Release October 2023 17
Specifying a Strict Sequence of Task Groups 56
Standard Port Flows 87
Starting with Bin in Port 124
State 18
Synchronizing with the Task Group Sequence 56
System Configuration 135
System Mode 144

T

Task 24
Task Group 23
Task Group Properties 23
Task Interface 13, 23
Task Interface and Bin Interface 12
Task Methods 38
Technical Interface 14, 140
Terms 113
Time 141
Time Factor 141
Transfer Bins to the other Grid 132

U

Update Bin 69, 35
Update Cell and Bin Content 114
Update Taskgroup 40
Using the AutoStore Interface 16

V

Value Codes 144

X

XML Message Format 15

AutoStore

AutoStore AS
Stokkastrandvegen 85
5578 Nedre Vats, NO
Tel: +47 52 76 35 00

AutoStore System Inc.
3 Corporate Park Drive
03038 Derry NH USA
Tel. +1 (603) 404-3479

