Adept ARCL Reference Guide

This is a PDF/print version of the Adept ARCL Reference Guide. A Table of Contents is provided so that you can locate the desired topics. Because the Adept ARCL Reference Guide was designed for online viewing, there may be slight formatting anomalies in the PDF/print version. Additionally, links to external documents will not work in the PDF file.

NOTE: Please see the ReadMe file, which is included with your Adept Motivity software, for a description of any recent changes.

Copyright Notice

Table Of Contents

Introduction to ARCL

The Advanced Robotics Command Language (ARCL) is a simple, text-based, command-and-response operating language for integrating a fleet of Adept mobile robots with an external automation system.

ARCL allows you to operate and monitor the mobile robot, its accessories and its payload devices over the network; it is intended for automating your mobile robots. For debugging purposes, you can use Telnet or PuTTY to access the ARCL commands from a command prompt.

ARCL allows you to submit jobs to the Enterprise Manager, and monitor the job status from start to finish. It also allows you to monitor payload information, if reported, by the robots in the fleet.

The Enterprise Manager (EM) version of ARCL is for use with the Enterprise Manager software and appliance. This hardware and software combination has been specially designed and configured to manage a fleet of robots operating in a facility. Therefore, it uses a minimal ARCL command set, because all of the critical work is being handled directly by the appliance and Enterprise Manager software.

This section discusses the following topics:

For more information on using the Adept Motivity software, refer to the *Adept Motivity Software User's Guide*.

See Also...

Introduction to ARCL on page 25
Enable Options in
Set ARCL Parameters in MobilePlanner on page 30
Connect to ARCL Using a Telnet Client on page 42
Using the ARCL Commands on page 46
ARCL Command Reference on page 70
ARCL Server Messages on page 299

Version Requirements

This document pertains to ARAM version 4.6 and later.

If you need assistance, see How Can I Get Help? on page 29.

See Also...

How Do I Begin on page 27

Related Manuals on page 28

How Can I Get Help? on page 29

How Do I Begin

Before you can access ARCL, you must complete the following steps:

1. Set ARCL Parameters in MobilePlanner

   Define the ARCL server address, port number and password parameters in MobilePlanner, and configure other ARCL parameters. The server port will not open without a password; therefore you must configure a password before you can connect to ARCL. For details, see Set ARCL Parameters in MobilePlanner on page 30.

2. Connect to ARCL Using a Telnet Client

   Using a Telnet client, connect to ARCL to access and run the ARCL commands on the Motivity platform. For details, see Connect to ARCL Using a Telnet Client on page 42.

After you've set up and established a connection to the ARCL server, you can start using the ARCL commands to operate and monitor the robotic platform, its accessories and its payload devices over the networksubmit and monitor jobs that will be performed by the fleet. You can do all of this with or without MobilePlanner. For more details, see Using the ARCL Commands on page 46.

See Also...

Version Requirements on page 26

How Do I Begin on page 27

Related Manuals on page 28

How Can I Get Help? on page 29

Related Manuals

In addition to this manual, you may want to refer to the following manuals which are available from the Adept Document Library.

Manual	Description
Adept Robot Safety Guide	Describes safety information for Adept robots.
Adept Motivity User's Guide	Describes the Adept Motivity software, including SetNetGo, MobileEyes, and MobilePlanner.
Adept Lynx Platform User's Guide	Describes the installation, start-up, operation, and maintenance of the Adept mobile robot base.
Adept Lynx Enterprise Manager 1100 User's Guide	Describes the installation and operation of the Enterprise Manager 1100 appliance and the Enterprise Manager software.
Adept SmartFleet EX Appliance User's Guide	Covers the legacy Adept Enterprise Manager system, which ran on an Adept SmartFleet EX Appliance, for managing a fleet of Lynx AIVs.

See Also...

Version Requirements on page 26

How Do I Begin on page 27

Related Manuals on page 28

How Can I Get Help? on page 29

How Can I Get Help?

For details on getting assistance with your Adept software or hardware, you can access the following information sources on the Adept corporate website:

- For contact information: http://www.adept.com/contact/americas
- For product support information: http://www.adept.com/support/service-and-support/main
- For user discussions, support, and programming examples: http://www.adept.com/forum/
- For further information about Omron Adept Technologies, Inc.: http://www.adept.com

See Also...

Version Requirements on page 26
How Do I Begin on page 27
Related Manuals on page 28
How Can I Get Help? on page 29

Set ARCL Parameters in MobilePlanner

This section describes how to access the configuration items in the MobilePlanner software. It describes the following:

- Accessing the Configuration Options on page 31
- Understanding the Configuration Parameters on page 37
- Outgoing ARCL Commands Parameters on page 39
- Set ARCL Parameters in MobilePlanner on page 30

Accessing the Configuration Options

These sections allow you to access configuration parameters that control the ARCL server and its interaction with connected clients.

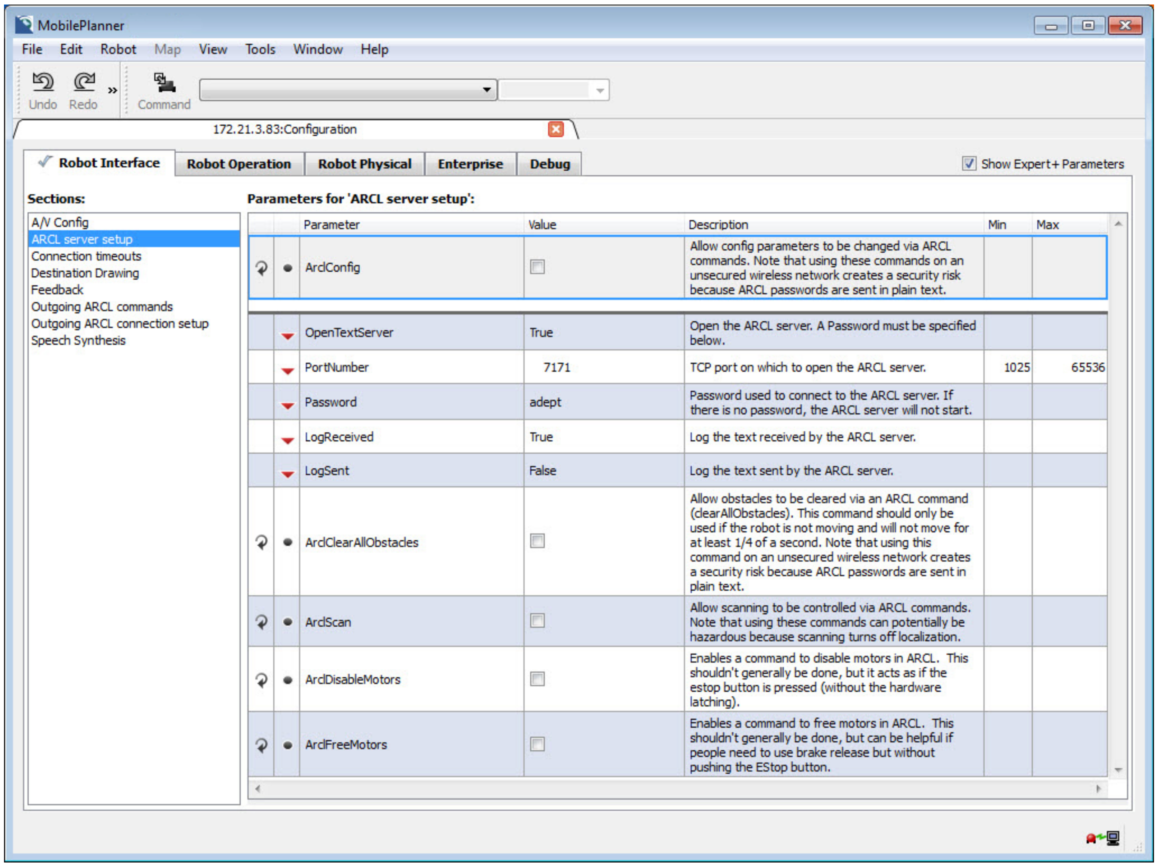
> **CAUTION:** The server port will not open without a password. Therefore, you must configure a password before you can connect to ARCL.

To access ARCL configuration options from MobilePlanner:

1. Open the MobilePlanner software, version 4.0 or later, and connect to the mobile robot. Refer to the *Adept Motivity User's Guide* for details on installing and starting MobilePlanner.
2. From the MobilePlanner > Config, select the Robot Interface tab.
3. Select ARCL server setup from the Sections: column. These parameters allow you to control the client-server connection between an offboard client process (such as Telnet or PuTTY) and ARCL. The ARCL server setup parameters are shown in the following figure.

   Incoming connections refer to a client initiating the connection to the Enterprise Manager or a robot. Multiple simultaneous connections are allowed and supported.

   > **NOTE**: ARCL server setup lets you configure the port for incoming connections. This does not affect outgoing connections.



Sections:

- A/V Config
- ARCL server setup
- Connection timeouts
- Destination Drawing
- Feedback
- Outgoing ARCL commands
- Outgoing ARCL connection setup
- Speech Synthesis

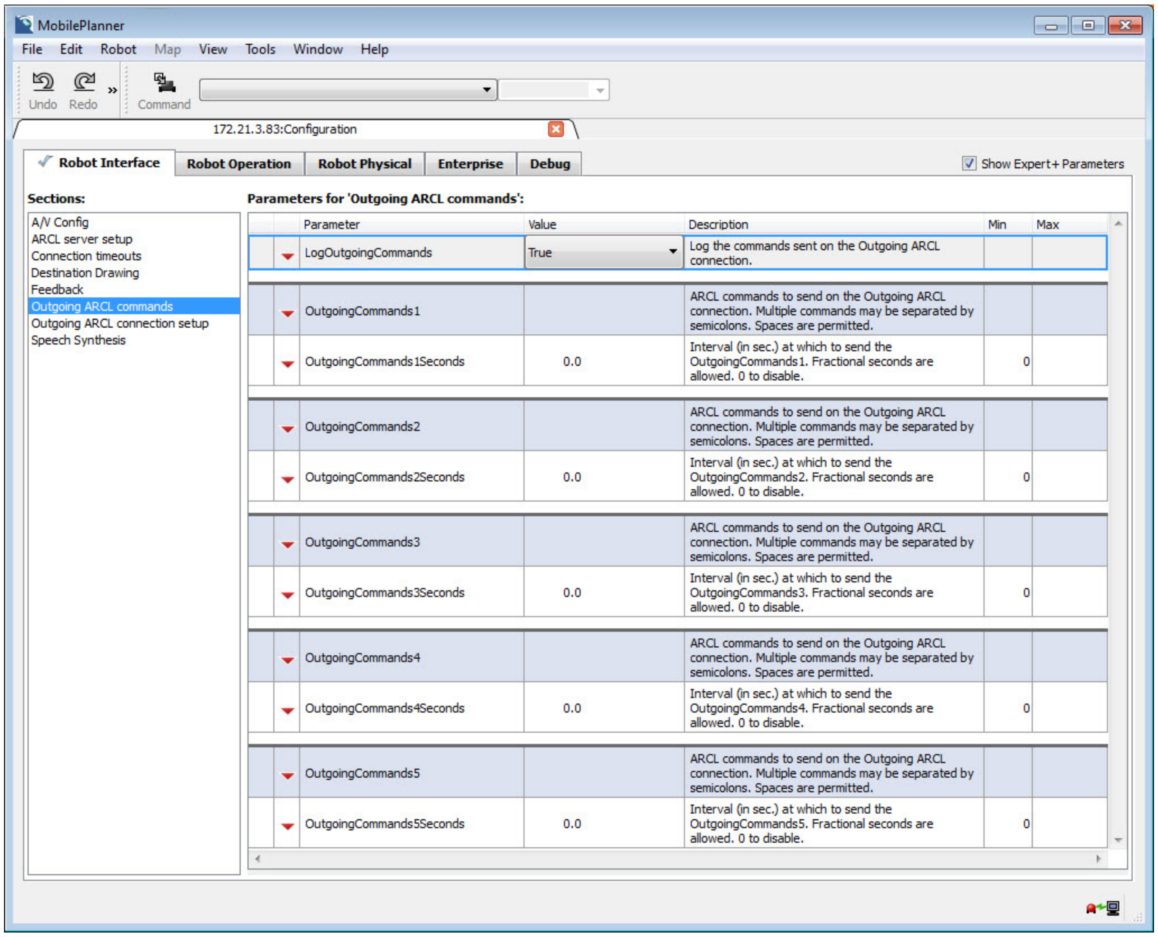
Parameters for 'ARCL server setup':

		Parameter	Value	Description	Min	Max
	●	ArclConfig	☐	Allow config parameters to be changed via ARCL commands. Note that using these commands on an unsecured wireless network creates a security risk because ARCL passwords are sent in plain text.		
	▼	OpenTextServer	True	Open the ARCL server. A Password must be specified below.		
	▼	PortNumber	7171	TCP port on which to open the ARCL server.	1025	65536
	▼	Password	adept	Password used to connect to the ARCL server. If there is no password, the ARCL server will not start.		
	▼	LogReceived	True	Log the text received by the ARCL server.		
	▼	LogSent	False	Log the text sent by the ARCL server.		
	●	ArclClearAllObstacles	☐	Allow obstacles to be cleared via an ARCL command (clearAllObstacles). This command should only be used if the robot is not moving and will not move for at least 1/4 of a second. Note that using this command on an unsecured wireless network creates a security risk because ARCL passwords are sent in plain text.		
	●	ArclScan	☐	Allow scanning to be controlled via ARCL commands. Note that using these commands can potentially be hazardous because scanning turns off localization.		
	●	ArclDisableMotors	☐	Enables a command to disable motors in ARCL. This shouldn't generally be done, but it acts as if the estop button is pressed (without the hardware latching).		
	●	ArclFreeMotors	☐	Enables a command to free motors in ARCL. This shouldn't generally be done, but can be helpful if people need to use brake release but without pushing the EStop button.		

ARCL Server Setup Parameters

For more information on using a client (like Telnet or PuTTY), see Connect to ARCL Using a Telnet Client on page 42.

4. Select Outgoing ARCL commands from the Sections: column to display the parameters that allow you to configure commands that are automatically executed on the connection indicated in the Outgoing ARCL connection setup. The parameters are shown in the following figure. For more details, see Outgoing ARCL Commands Parameters on page 39.



Sections:

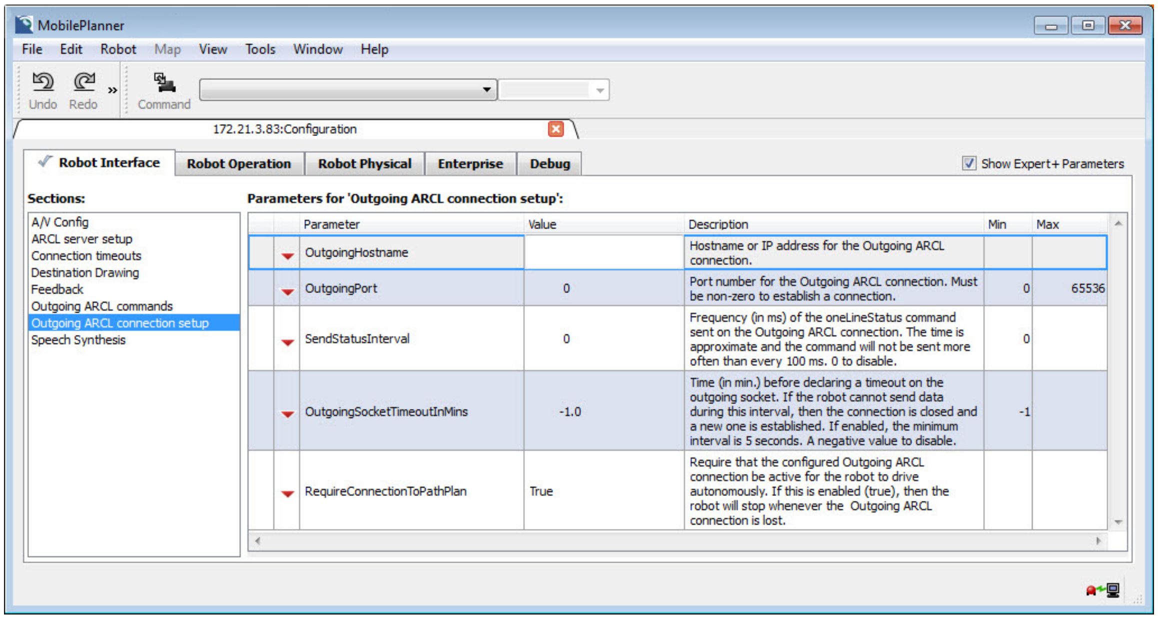
- A/V Config
- ARCL server setup
- Connection timeouts
- Destination Drawing
- Feedback
- Outgoing ARCL commands
- Outgoing ARCL connection setup
- Speech Synthesis

Parameters for 'Outgoing ARCL commands':

Parameter	Value	Description	Min	Max
LogOutgoingCommands	True	Log the commands sent on the Outgoing ARCL connection.		
OutgoingCommands1		ARCL commands to send on the Outgoing ARCL connection. Multiple commands may be separated by semicolons. Spaces are permitted.		
OutgoingCommands1Seconds	0.0	Interval (in sec.) at which to send the OutgoingCommands1. Fractional seconds are allowed. 0 to disable.	0	
OutgoingCommands2		ARCL commands to send on the Outgoing ARCL connection. Multiple commands may be separated by semicolons. Spaces are permitted.		
OutgoingCommands2Seconds	0.0	Interval (in sec.) at which to send the OutgoingCommands2. Fractional seconds are allowed. 0 to disable.	0	
OutgoingCommands3		ARCL commands to send on the Outgoing ARCL connection. Multiple commands may be separated by semicolons. Spaces are permitted.		
OutgoingCommands3Seconds	0.0	Interval (in sec.) at which to send the OutgoingCommands3. Fractional seconds are allowed. 0 to disable.	0	
OutgoingCommands4		ARCL commands to send on the Outgoing ARCL connection. Multiple commands may be separated by semicolons. Spaces are permitted.		
OutgoingCommands4Seconds	0.0	Interval (in sec.) at which to send the OutgoingCommands4. Fractional seconds are allowed. 0 to disable.	0	
OutgoingCommands5		ARCL commands to send on the Outgoing ARCL connection. Multiple commands may be separated by semicolons. Spaces are permitted.		
OutgoingCommands5Seconds	0.0	Interval (in sec.) at which to send the OutgoingCommands5. Fractional seconds are allowed. 0 to disable.	0	

Outgoing ARCL Commands

5. Select Outgoing ARCL connection setup from the Sections: column to display the parameters that allow you to send data from the robot using ARCL commands, intended to connect to the application payload. The parameters are shown in the following figure. For more details, refer to Outgoing ARCL Connection Setup Parameters on page 38.



Sections:

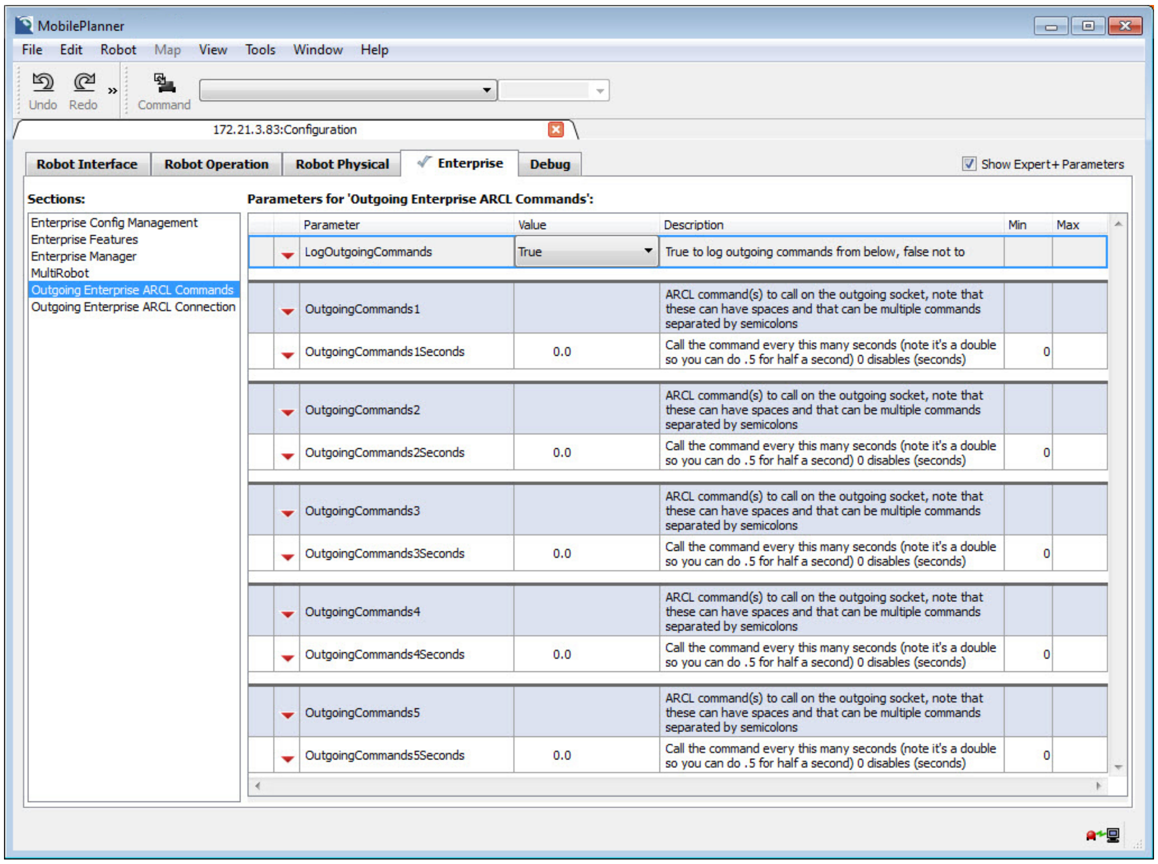
- A/V Config
- ARCL server setup
- Connection timeouts
- Destination Drawing
- Feedback
- Outgoing ARCL commands
- Outgoing ARCL connection setup
- Speech Synthesis

Parameters for 'Outgoing ARCL connection setup':

Parameter	Value	Description	Min	Max
OutgoingHostname		Hostname or IP address for the Outgoing ARCL connection.		
OutgoingPort	0	Port number for the Outgoing ARCL connection. Must be non-zero to establish a connection.	0	65536
SendStatusInterval	0	Frequency (in ms) of the oneLineStatus command sent on the Outgoing ARCL connection. The time is approximate and the command will not be sent more often than every 100 ms. 0 to disable.	0	
OutgoingSocketTimeoutInMins	-1.0	Time (in min.) before declaring a timeout on the outgoing socket. If the robot cannot send data during this interval, then the connection is closed and a new one is established. If enabled, the minimum interval is 5 seconds. A negative value to disable.	-1	
RequireConnectionToPathPlan	True	Require that the configured Outgoing ARCL connection be active for the robot to drive autonomously. If this is enabled (true), then the robot will stop whenever the Outgoing ARCL connection is lost.		

Outgoing ARCL Connection Setup

6. After the configuration options are set, click the Save button on the toolbar to save the changes to the Configuration file. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.
7. Select Outgoing Enterprise ARCL commands from the Sections: column to display the parameters that allow you to configure commands that are automatically executed on the connection indicated in the Outgoing Enterprise ARCL connection setup. For more details, see Outgoing Enterprise ARCL Commands Parameters on page 41.



Sections:

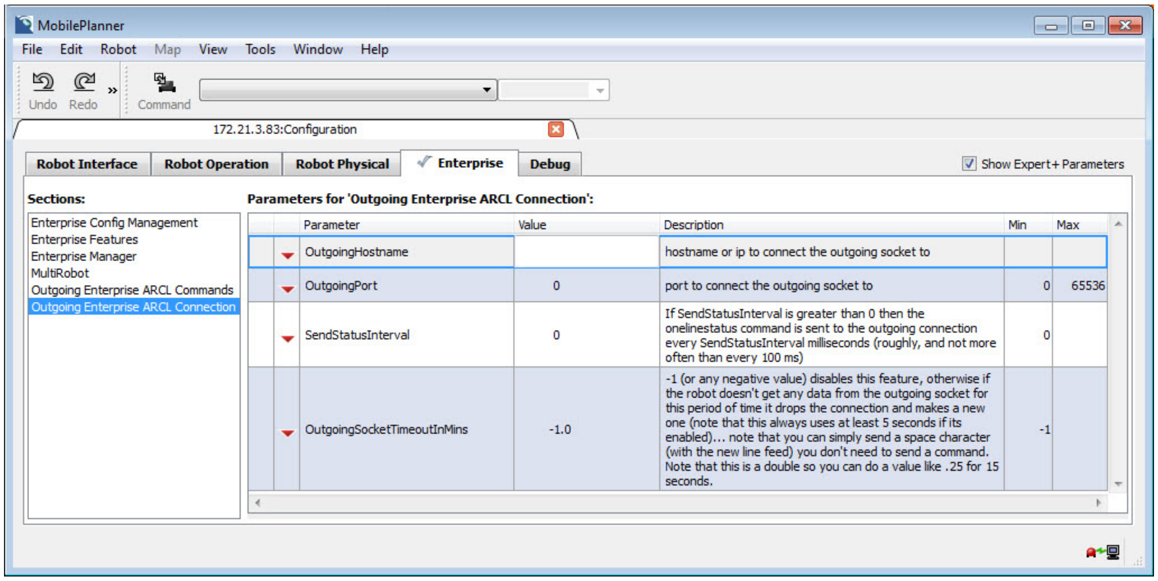
- Enterprise Config Management
- Enterprise Features
- Enterprise Manager
- MultiRobot
- Outgoing Enterprise ARCL Commands
- Outgoing Enterprise ARCL Connection

Parameters for 'Outgoing Enterprise ARCL Commands':

Parameter	Value	Description	Min	Max
LogOutgoingCommands	True	True to log outgoing commands from below, false not to		
OutgoingCommands1		ARCL command(s) to call on the outgoing socket, note that these can have spaces and that can be multiple commands separated by semicolons		
OutgoingCommands1Seconds	0.0	Call the command every this many seconds (note it's a double so you can do .5 for half a second) 0 disables (seconds)	0	
OutgoingCommands2		ARCL command(s) to call on the outgoing socket, note that these can have spaces and that can be multiple commands separated by semicolons		
OutgoingCommands2Seconds	0.0	Call the command every this many seconds (note it's a double so you can do .5 for half a second) 0 disables (seconds)	0	
OutgoingCommands3		ARCL command(s) to call on the outgoing socket, note that these can have spaces and that can be multiple commands separated by semicolons		
OutgoingCommands3Seconds	0.0	Call the command every this many seconds (note it's a double so you can do .5 for half a second) 0 disables (seconds)	0	
OutgoingCommands4		ARCL command(s) to call on the outgoing socket, note that these can have spaces and that can be multiple commands separated by semicolons		
OutgoingCommands4Seconds	0.0	Call the command every this many seconds (note it's a double so you can do .5 for half a second) 0 disables (seconds)	0	
OutgoingCommands5		ARCL command(s) to call on the outgoing socket, note that these can have spaces and that can be multiple commands separated by semicolons		
OutgoingCommands5Seconds	0.0	Call the command every this many seconds (note it's a double so you can do .5 for half a second) 0 disables (seconds)	0	

Outgoing Enterprise ARCL Commands

8. Select Outgoing Enterprise ARCL connection setup from the Sections: column to display the parameters that allow you to send data from the Enterprise Manager using ARCL commands, intended to connect to the facility WMS/MES. For more details, refer to Outgoing Enterprise ARCL Connection Setup Parameters on page 40.



Sections:

- Enterprise Config Management
- Enterprise Features
- Enterprise Manager
- MultiRobot
- Outgoing Enterprise ARCL Commands
- Outgoing Enterprise ARCL Connection

Parameters for 'Outgoing Enterprise ARCL Connection':

Parameter	Value	Description	Min	Max
OutgoingHostname		hostname or ip to connect the outgoing socket to		
OutgoingPort	0	port to connect the outgoing socket to	0	65536
SendStatusInterval	0	If SendStatusInterval is greater than 0 then the onelinestatus command is sent to the outgoing connection every SendStatusInterval milliseconds (roughly, and not more often than every 100 ms)	0	
OutgoingSocketTimeoutInMins	-1.0	-1 (or any negative value) disables this feature, otherwise if the robot doesn't get any data from the outgoing socket for this period of time it drops the connection and makes a new one (note that this always uses at least 5 seconds if its enabled)... note that you can simply send a space character (with the new line feed) you don't need to send a command. Note that this is a double so you can do a value like .25 for 15 seconds.	-1	

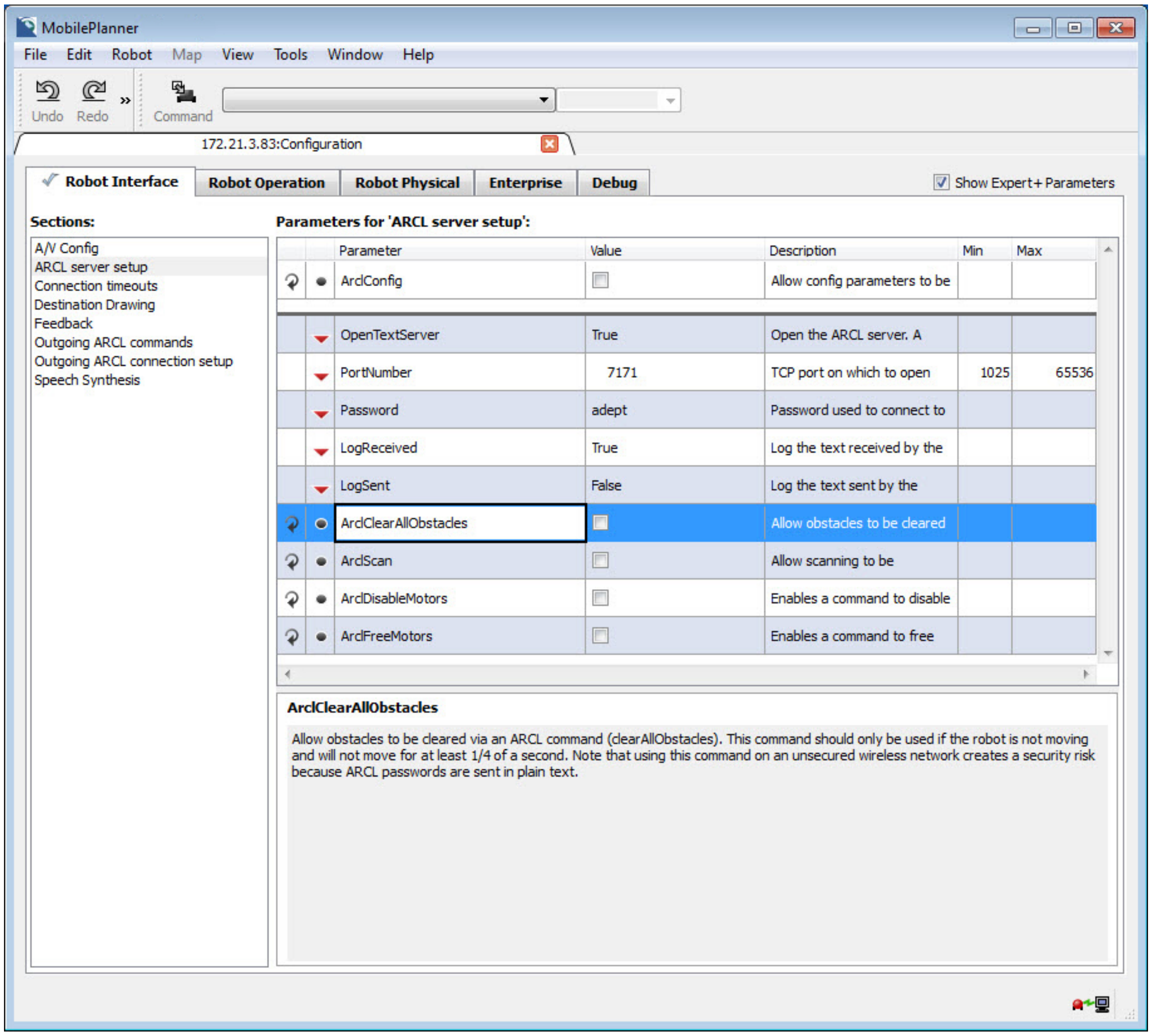
Outgoing Enterprise ARCL Connection Setup

9. After the configuration options are set, click the Save button on the toolbar to save the changes to the Configuration file. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Understanding the Configuration Parameters

The configuration parameters are grouped by function - each functional group is accessed from the alphabetical list in the left pane. The corresponding configuration parameters are listed in a tabular format on the configuration pages, as shown in the previous figures. The parameters are organized alphabetically. You can sort the list in ascending or descending order by name, value, min, or max.

Each parameter has a description that briefly describes the function of the parameter. The selected parameter's help description is located in the Description column and, optionally, at the bottom of the window when the entire contents can't be shown in the Description column. For an example, see the following figure.



Sections:

- A/V Config
- ARCL server setup
- Connection timeouts
- Destination Drawing
- Feedback
- Outgoing ARCL commands
- Outgoing ARCL connection setup
- Speech Synthesis

Parameters for 'ARCL server setup':

Parameter	Value	Description	Min	Max
ArclConfig	☐	Allow config parameters to be		
OpenTextServer	True	Open the ARCL server. A		
PortNumber	7171	TCP port on which to open	1025	65536
Password	adept	Password used to connect to		
LogReceived	True	Log the text received by the		
LogSent	False	Log the text sent by the		
ArclClearAllObstacles	☐	Allow obstacles to be cleared		
ArclScan	☐	Allow scanning to be		
ArclDisableMotors	☐	Enables a command to disable		
ArclFreeMotors	☐	Enables a command to free		

ArclClearAllObstacles

Allow obstacles to be cleared via an ARCL command (clearAllObstacles). This command should only be used if the robot is not moving and will not move for at least 1/4 of a second. Note that using this command on an unsecured wireless network creates a security risk because ARCL passwords are sent in plain text.

Parameter Help

Outgoing ARCL Connection Setup Parameters

The Outgoing ARCL connection setup parameters are used to instruct the AIV to initiate an outgoing ARCL TCP connection to another device on the network. This approach can be used in lieu of requiring that the other device initiate an incoming connection to the AIV.

In order to use this feature, the OutgoingHostname needs to be set to a string and the OutgoingPort needs to be a non-zero number.

Use of the outgoing ARCL connections:

- The outgoing ARCL connection can be used to connect to a payload on top of the AIV. The AIV can be configured so that it will not autonomously drive unless the outgoing connection is alive, by setting the Outgoing ARCL Connection setup -> RequireConnectionToPathPlan parameter to True.

  This is useful when it would be unsafe for the AIV to move at certain times, such as when an automated load or unload is being performed. The payload is responsible for signaling when it is safe to move, so if the connection from the payload to the AIV is lost, it would be unsafe for the AIV to move without knowing the payload status.

  There may be hand-shaking involved between the AIV's payload and the factory equipment, to determine when the load or unload is complete, making it safe for the AIV to move.

- The outgoing connection can be used to automatically execute certain ARCL commands at specified intervals. This can be useful for gathering certain information without requiring that the application, running on the connected device, continuously request the data.

Outgoing ARCL Commands Parameters

The Outgoing ARCL command parameters allow you to set the mobile robot up to automatically generate ARCL commands at regular intervals. You can send one or more ARCL commands; to send multiple commands, separate each command with a pipe charactger (|). For example, set the OutGoingCommands1 parameter to:

```
doTaskInstant sayInstant "Enabling motors." | enableMotors
```

Then set the OutGoingCommands1Seconds parameter to:

```
60
```

Every 60 seconds, the mobile robot will announce, "Enabling motors" and then attempt to enable the motors.

The outgoing host will receive the ARCL responses:

```
Completed doing instant task: sayInstant "Enabling motors."
```

Then it will respond with, either:

```
Motors enabled
```

or

```
Estop pressed, cannot enable motors
```

Outgoing Enterprise ARCL Connection Setup Parameters

The Outgoing Enterprise ARCL connection setup parameters are used to instruct the Enterprise Manager to initiate an outgoing ARCL TCP connection to another device on the network. This approach can be used in lieu of requiring that the other device initiate an incoming connection to the Enterprise Manager.

There may be hand-shaking involved between the Enterprise Manager and the factory equipment, to determine when the command should be executed.

In order to use this feature, the OutgoingHostname needs to be set to a string and the OutgoingPort needs to be a non-zero number.

Use of the outgoing ARCL connections:

- The outgoing connection can be used to automatically execute certain ARCL commands at specified intervals. This can be useful for gathering certain information without requiring that the application, running on the connected device, continuously request the data.

Outgoing Enterprise ARCL Commands Parameters

The Outgoing Enterprise ARCL command parameters allow you to set the Enterprise Manager up to automatically generate ARCL commands at regular intervals. You can send one or more ARCL commands; to send multiple commands, separate each command with a pipe character (|). For example, set the OutGoingCommands1 parameter to:

```
Queueshowrobot default echoit

QueueRobot: “Robot1” UnAvailable EStopPressed echoit
QueueRobot: “Robot2” UnAvailable Interrupted echoit
QueueRobot: “Robot3” UnAvailable InterruptedButNotYetIdle echoit
QueueRobot: “Robot4” Available Available echoit
QueueRobot: “Robot5” InProgress Driving  echoit
QueueRobot: “Robot6” UnAvailable NotUsingEnterpriseManager echoit
QueueRobot: “Robot7” UnAvailable UnknownBatteryType echoit
QueueRobot: “Robot8” UnAvailable ForcedDocked echoit
QueueRobot: “Robot9” UnAvailable NotLocalized echoit
QueueRobot: "patrolbot" UnAvailable Fault_Driving_Application_faultName echoit

EndQueueShowRobot
```

Then you could parse the output to compare the number of robots connected vs. how many robots should be connected, and generate an alarm if there is a mismatch.

See Also...

Introduction to ARCL on page 25

Enable Options in

Set ARCL Parameters in MobilePlanner on page 30

Connect to ARCL Using a Telnet Client on page 42

Using the ARCL Commands on page 46

ARCL Command Reference on page 70

ARCL Server Messages on page 299

Connect to ARCL Using a Telnet Client

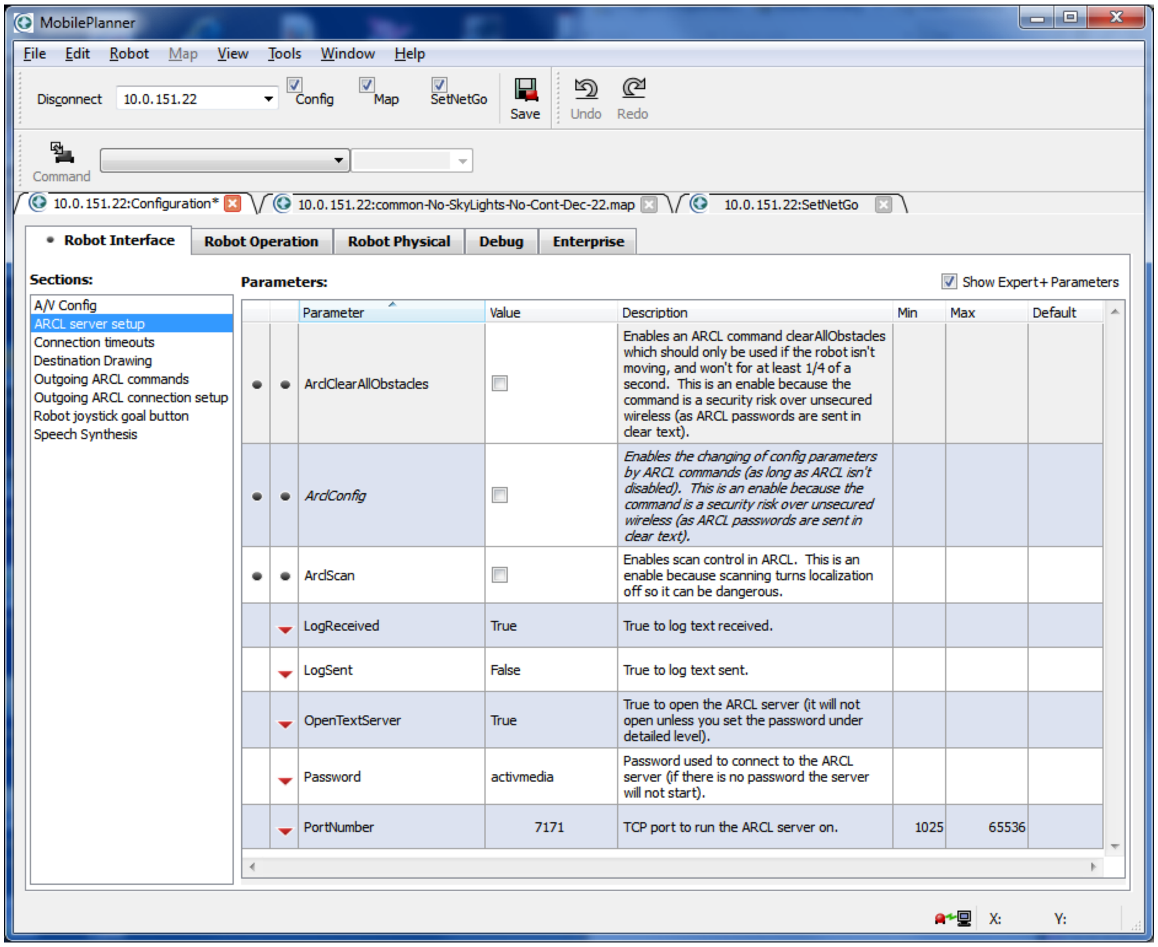
This section tells you how to connect to your mobile robot to ARCL using a client, such as Telnet or PuTTY.

Setting the Connection Parameters

1. Open the MobilePlanner software, version 4.0 or later, and connect to the mobile robot. Refer to the *Adept Motivity User's Guide* for details on installing and starting MobilePlanner.

2. From the Configuration tab, select the Robot Interface tab.

3. Select ARCL Server Setup from the Sections column. The ARCL Server Setup parameters are shown in the following figure.


MobilePlanner
File Edit Robot Map View Tools Window Help
Disconnect 10.0.151.22 Config Map SetNetGo Save Undo Redo
Command
10.0.151.22:Configuration*
10.0.151.22:common-No-SkyLights-No-Cont-Dec-22.map
10.0.151.22:SetNetGo
Robot Interface Robot Operation Robot Physical Debug Enterprise
Show Expert+ Parameters
X: Y:

Sections:

- A/V Config
- ARCL server setup
- Connection timeouts
- Destination Drawing
- Outgoing ARCL commands
- Outgoing ARCL connection setup
- Robot joystick goal button
- Speech Synthesis

Parameters:

Parameter	Value	Description	Min	Max	Default
ArclClearAllObstacles	☐	Enables an ARCL command clearAllObstacles which should only be used if the robot isn't moving, and won't for at least 1/4 of a second. This is an enable because the command is a security risk over unsecured wireless (as ARCL passwords are sent in clear text).			
ArclConfig	☐	*Enables the changing of config parameters by ARCL commands (as long as ARCL isn't disabled). This is an enable because the command is a security risk over unsecured wireless (as ARCL passwords are sent in clear text).*			
ArclScan	☐	Enables scan control in ARCL. This is an enable because scanning turns localization off so it can be dangerous.			
LogReceived	True	True to log text received.			
LogSent	False	True to log text sent.			
OpenTextServer	True	True to open the ARCL server (it will not open unless you set the password under detailed level).			
Password	activmedia	Password used to connect to the ARCL server (if there is no password the server will not start).			
PortNumber	7171	TCP port to run the ARCL server on.	1025	65536	

ARCL Server Setup Parameters

These parameters allow you to control the client-server connection, see Understanding the Configuration Parameters on page 37 for details.

4. Enter a password for the Telnet client for the Password parameter. If a password already exists, make a note of it so that you can open the ARCL server from the Telnet connection.

Connecting to ARCL

The following instructions describe how to connect to ARCL using the Command Prompt window in the Microsoft Windows operating system. You can also use a terminal-emulation utility, such as PuTTY. For details on PuTTY, see the PuTTY website: http://www.putty.org.

1. On a Windows-based PC, open the Command Prompt window.

   In Windows, hold down the "Window" key and the "R" key to open the Run dialog box. Type **cmd** to display the command terminal.)

   > **NOTE:** On some Windows installations, you may need to enable Telnet using:
   >
   > Control Panel > Programs and Features > Turn Windows feature on or off.

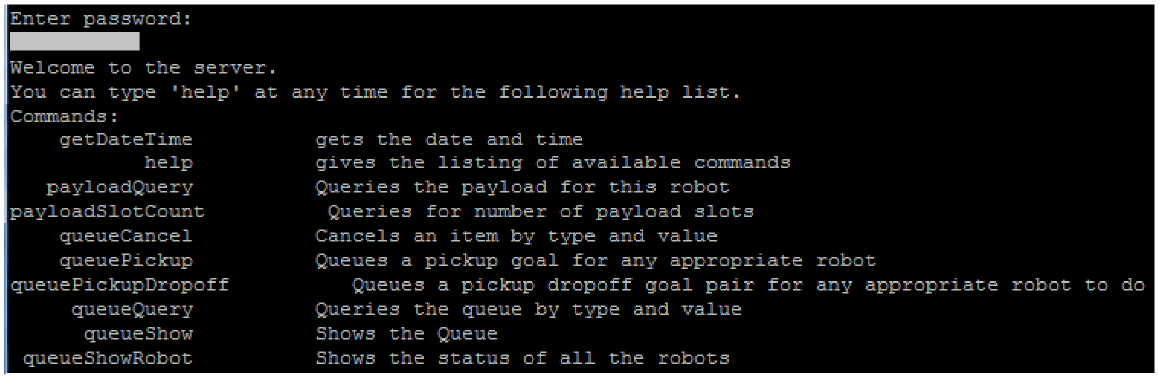
2. Start Telnet using the ARCL server address and the port number specified in the ARCL Server Setup Parameters. For example:

   ```
   Telnet 192.168.0.44 7171
   ```

3. Enter the password that you set in Step 5, above. If you mis-type the password, you will have to restart the Telnet client.

   After you have successfully logged-in, the server responds with a list of supported commands and a brief description of each. See the example in the following figure.

   > **NOTE:** The list of available commands depends on your system configuration.



```
Enter password:
[redacted]
Welcome to the server.
You can type 'help' at any time for the following help list.
Commands:
    getDateTime          gets the date and time
           help          gives the listing of available commands
   payloadQuery          Queries the payload for this robot
payloadSlotCount          Queries for number of payload slots
    queueCancel          Cancels an item by type and value
    queuePickup          Queues a pickup goal for any appropriate robot
queuePickupDropoff          Queues a pickup dropoff goal pair for any appropriate robot to do
     queueQuery          Queries the queue by type and value
      queueShow          Shows the Queue
 queueShowRobot          Shows the status of all the robots
```

Example Command List after Login

4. If needed, you can enter the **echo off** command to prevent your input from echoing (typing double characters).

5. When you are finished, use the **quit** command to properly close the connection.

After you connect to ARCL, you can execute any of the ARCL commands available. For a complete list of the different ARCL commands and their arguments, refer to ARCL Command Reference on page 70.

ARCL supports multiple client/server connections through the TCP/IP socket. However, commands and query responses are connection-specific. For example, you can have two Telnet clients connected; however, only the one that requested a **oneLineStatus** response actually receives the status message.

See Also...

Introduction to ARCL on page 25

Enable Options in

Set ARCL Parameters in MobilePlanner on page 30

Connect to ARCL Using a Telnet Client on page 42

Using the ARCL Commands on page 46

ARCL Command Reference on page 70

ARCL Server Messages on page 299

Using the ARCL Commands

After you have established a connection to the ARCL server, you are ready to operate and monitor the mobile robot using the ARCL commands. The following topics discuss the use of these commands for certain tasks. To view an alphabetical list and description of each ARCL command, refer to ARCL Command Reference on page 70.

This section discusses the following topics:

The ARCL command set is evolutionary and backward compatible. To see added commands, consult the ARCL help list when connecting with a new ARAM version. For more details on the help command, see help Command on page 141.

See Also...

Introduction to ARCL on page 25

Enable Options in

Set ARCL Parameters in MobilePlanner on page 30

Connect to ARCL Using a Telnet Client on page 42

Using the ARCL Commands on page 46

ARCL Command Reference on page 70

ARCL Server Messages on page 299

Understanding the Commands

This section describes the document conventions, command notes, and status and error messages.

The commands are discussed by task in this chapter. To view commands presented in alphabetical order, see the ARCL Command Reference on page 70.

Document Conventions

Command name (shortcut: cn)

The command can be invoked with its full name or, in some cases, with a shortcut. When there is a shortcut, it will be listed in parentheses after the command name in the title of the command description. The syntax, usage, and parameters are the same, whether the full command name or the shortcut is used.

Syntax

The ARCL commands are not case sensitive. In this guide, commands are shown in mixed case and bold type. Required parameters are shown in angled brackets and regular type; whereas, optional parameters are shown in square brackets [ ] and regular type. For example:

queuePickup <goalName> [priority] [jobId]

In this example, the <goalName> parameter is required; the [priority] and [jobId] parameters are optional.

goToRouteGoal <routeName> <goalName> [index]

In this example, the <routeName> and <goalName> parameters are required; the [index] parameter is optional.

Usage Considerations

This section describes any special considerations that must be followed when using the command. It also describes where the command can be used, as follows:

- This ARCL command is only available on the robot.
- This ARCL command is available only on the Enterprise Manager.
- This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

This section lists any ARAM settings that must be enabled to use the command.

Parameters

This section describes each of the required and optional command parameters (such as goalname, routname, echo, etc.).

Responses

This section shows the information returned by the command.

Details

This section provides more details about the functions of the command.

Examples

This section provides examples of correctly-formatted command lines are presented in this section.

Related Commands

This section lists additional commands that are similar or often used with this command.

Command Notes

Below are some helpful notes to remember when using ARCL commands:

- ARCL responds with the command’s syntax if you omit any or all required parameters.
- Extraneous parameters are ignored.
- ARCL limits commands to a maximum of 5,000 ASCII characters
- As a general rule, use double quotes for string parameters, especially if there are spaces in the string.
- Mistyped Telnet commands and parameters cannot be edited on the command line. You have to completely re-type the command.
- Mistyped or non-existent commands are rejected with the response, “Unknown command".
- Although commands are not case-sensitive, some parameters are case-sensitive.

Data Types

The following table shows all the available ARCL data types (not all of these may apply to a particular command):

Parameter	Data Type	Max Length/Range
cancelType	string	max length: 127 characters
cancelValue	string	max length: 127 characters
DROPOFFgoalName	string	max length: 127 characters
DROPOFFpriority	integer (signed long)	range: −2147483648 to 2147483647
echoString[2]	string	max length: 127 characters
goalName	string	max length: 127 characters
jobId[2]	string	max length: 127 characters
payload slot number	integer (signed long)	range: 1 to 2147483647
payload slot string[1]	string	max length: 127 characters
PICKUPgoalName	string	max length: 127 characters
PICKUPpriority	integer (signed long)	range: −2147483648 to 2147483647
priority	integer (signed long)	range: −2147483648 to 2147483647
queryType	string	max length: 127 characters
queryValue	string	max length: 127 characters
reason[2]	string	max length: 127 characters
robotName[1]	string	max length: 127 characters
[1]These parameters support spaces, and need to be enclosed in quotes if they include spaces.		
[2]These parameters do not support spaces or double quotes.		

Status and Error Messages

ARCL sends important status updates to the connected client for certain commands, such as goto goalName. For example, when the mobile robot first starts toward the goal, the following is sent to the client:

```
Going   to   goal goalName
```

When the robot arrives at the goal, a status update of the following is displayed:

```
Arrived   at   goal goalName
```

If ARCL is unable to execute the command because of a command sequence error, a non-existent filename, or because a feature was not set up properly, a SetUpError is displayed. For example, if you attempt to execute listAdd or listExecute before entering the command listStart, the following error is displayed:

```
SetUpError: You need to start a list before you can add to it.
```

All other argument errors result in a two-line ARCL response, with two distinct error messages, such as the following:

```
CommandError:   goto   dock12
CommandErrorDescription:   No  goal   'dock12'
```

Occasionally, ARCL sends reports without prompting, for example, when there are changes in the robot’s docking and charging status.

ARCL sends important status updates to the connected client for certain commands, such as **queuePickup** goalName. For example, when the job is first received, then the following is sent to the client:

```
queuepickup goal "<goalName>” with priority 10, id PICKUP138 and jobId JOB138
successfully queued
```

When the job has been completed, this update message is sent:

```
QueueUpdate: PICKUP138 JOB138 10 Completed None Goal "<goalName>" "robotName"
04/08/2013 13:46:34 0
```

If ARCL is unable to execute the command because of a command sequence error, a non-existent filename, or because a feature was not set up properly, a SetUpError is displayed. For example, if you attempt to execute listAdd or listExecute before entering the command listStart, the following error is displayed:

```
SetUpError: You need to start a list before you can add to it.
```

All other argument errors result in a two-line ARCL response, with two distinct error messages, such as the following:

```
CommandError: queuePickup goal6
CommandErrorDescription: queuePickup no such goal "goal6"
```

ARCL sends status update messages without prompting, for example, when there are changes in a robot’s or a job's state.

Refer to ARCL Server Messages on page 299 for a list of unprompted messages.

Status Conditions

The following table shows the possible robot and job status conditions:

Status	Substatus
Pending	None
Pending	AssignedRobotOffLine
Pending	NoMatchingRobotForLinkedJob
Pending	NoMatch-ingRobotForOtherSegment
Pending	NoMatchingRobot
Pending	ID_PICKUPxx <where PICKUPxx is the jobSegment ID for which this Job Segment is waiting>
Pending	ID_DROPOFFxx <where DROPOFFxx is the jobSegment ID for which this Job Segment is waiting>
Available	Available
Available	Parking
Available	Parked
Available	DockParking
Available	DockParked
Interrupted	None
InProgress	UnAllocated
InProgress	Allocated
InProgress	BeforePickup
InProgress	BeforeDropoff
InProgress	BeforeEvery
InProgress	Before
InProgress	Buffering
InProgress	Buffered
InProgress	Driving
InProgress	After
InProgress	AfterEvery

Status	Substatus
InProgress	AfterPickup
InProgress	AfterDropoff
Completed	None
Cancelling	None
Cancelled	None
Cancelling	<application_supplied_can-celReason_string>
Cancelled	<application_supplied_can-celReason_string>
BeforeModify	None
Inter-ruptedByModify	None
AfterModify	None
UnAvailable	NotUsingEnterpriseManager
UnAvailable	UnknownBatteryType
UnAvailable	ForcedDocked
UnAvailable	Lost
UnAvailable	EStopPressed
UnAvailable	Interrupted
UnAvailable	InterruptedButNotYetIdle
UnAvailable	Fault_Driving_Application_ <application_supplied_string>
UnAvailable	OutgoingARCLConnLost
UnAvailable	Parking
UnAvailable	DockParking
UnAvailable	ModeIsLocked

See Also...

Understanding the Commands on page 47

Using ARCL Variables on page 55

Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Using ARCL Variables

The following is a list of variables that you can use with any ARCL command.

Variable	Description/Range of Values
$g	Represents the current goal name. For example: Going to goal $g.
$y	Represents the year (2xxx)
$m	Represents the month (1-12)
$d	Represents the day (1-7)
$H	Represents the hour (0-23)
$M	Represents the minute (0-59)
$S	Represents the second (0-59)
$T	Represents the current heading (Th) of the mobile robot (degrees)
$X	Represents the current X position of the mobile robot in the map (mm)
$Y	Represents the current Y position of the mobile robot in the map (mm)

See Also...

Understanding the Commands on page 47
Using ARCL Variables on page 55
Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Using Tasks and Macros

ARCL's list commands let you assemble and execute a sequence of tasks, or execute macros. The following ARCL commands allow you to carry out a single task, create a task list, or execute a macro:

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

play Command on page 199

say Command on page 261

ARCL also allows you to create a task list, add tasks to the list, and then execute the task list. In doing so, you can make use of the tasks that are available in the MobilePlanner software for building routes and macros. Refer to the list commands for details.

Initialize a list first with the listStart command. This also overwrites a list that you may've already started, but have yet to execute. Use the listAdd command with a task name as argument and followed by any and all task arguments to put a task into the current list. Use listExecute to perform the series of tasks, each in order first in, first out. The list may be executed only once, and you must start with listStart before creating and executing a new list sequence of tasks.

Here is a simple example that has the robot travel to the goal Lobby and, when it gets there, says its name and then asks for your name. The regular text lines are what you might type; the lines in quotes are the messages that the ARCL server generates as you send it commands and it completes its tasks:

```
listStart
"List being cleared"
"Making new list"
listAdd goto Lobby
"Added task goto Lobby"
listAdd say My name is PatrolBot.
"Added task say My name is PatrolBot."
listAdd say What is your name?
"Added task say What is your name?"
listExecute
"Executing list"
"Successfully finished task list"
```

To carry out a single task use the doTask command. For example:

```
doTask goto goal_name
```

Forever Tasks

There are a few tasks in the MobilePlanner software that end with the qualifier "Forever". This means that the task continues until explicitly instructed to do something else. The patrolForever command, for example, causes the robot to continuously patrol the specified route. In other words, it keeps repeating the route until it is commanded to stop.

Therefore, it is best to avoid using "Forever" robot tasks in a task list or with the doTask command in ARCL. Instead use the dock or patrol ARCL commands, which serve the same purpose. The differences are subtle, but the dock and patrol commands are more appropriate for the job.

See Also...

Understanding the Commands on page 47

Using ARCL Variables on page 55

Using Tasks and Macros on page 56

Using Configuration Commands on page 58

Using the Queuing Commands on page 60

Working With Payloads on page 61

Creating a Map on page 63

Tracking Sectors on page 64

Navigating and Localizing on page 65

Using Range Devices and Custom Sensors on page 66

Monitoring the I/O Ports on page 68

Using Configuration Commands

ARCL allows you change the value of one or more ARAM operating parameters. For example, you can tell it to use a different map or change its top speed while driving. The following configuration commands are supported:

configAdd Command on page 85

configParse Command on page 87

configStart Command on page 89

getConfigSectionInfo Command on page 117

getConfigSectionList Command on page 119

getConfigSectionValues Command on page 121

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

NOTE: You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Use the configStart command to initialize a configuration list, similar to creating a task list. The configStart command overwrites any previous list. Use the configAdd command to enter sections and related configuration parameter keywords and values to the list. The configParse command sends the configuration parameters to ARAM, which implements the configuration changes.

When creating the configuration list, you must first identify which Section the configuration parameter(s) is/are associated, and then provide the parameter's keyword and new value. Configuration keywords are case-sensitive. For example, to change to a different map on the robot:

```
configStart
New config starting
configAdd Section Files
Added 'Section Files' to the config
configAdd Map theNewMap.map
Added 'Map theNewMap.map' to the config
configParse
Will parse config
Map changed
Config parsed fine
```

Notice that the "Map changed" response was not generated by ARCL, but rather is an ARAM event warning that is sent automatically to all attached clients. See ARCL Server Messages on page 299 for details. ARAM catches and reports errors both for configuration and system issues, for example if it is unable to find a file or correctly load a map file.

To view ARAM configuration details and parameter values, use the ARCL commands: getConfigSectionList, getConfigSectionValues, and getConfigSectionInfo.

You can also create and manage custom configuration sections and parameters from ARCL. These new sections and parameters are saved into a downloaded configuration file. However, new sections and parameters do not persist, even if recently uploaded from a saved configuration file. Instead, you must execute the newConfigParam command whenever restarting ARAM. However, the last value given to the parameter persists.

See Also...

Understanding the Commands on page 47

Using ARCL Variables on page 55

Using Tasks and Macros on page 56

Using Configuration Commands on page 58

Using the Queuing Commands on page 60

Working With Payloads on page 61

Creating a Map on page 63

Tracking Sectors on page 64

Navigating and Localizing on page 65

Using Range Devices and Custom Sensors on page 66

Monitoring the I/O Ports on page 68

'

Using the Queuing Commands

The ARCL queuing commands are used with the Enterprise Manager. They allow you to request a mobile robot to drive to a goal (for example, for a pickup) and then drive to another goal (for example, for a dropoff).

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queueModify Command (shortcut: qmod) on page 218

queueModify Command (shortcut: qmod) on page 218

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

See Also...

Understanding the Commands on page 47
Using ARCL Variables on page 55
Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Working With Payloads

Using the ARCL payload commands, you can view the number of slots on a robot, assign names to those slot numbers, define the object (or payload) you want the robot to pick up or drop off, see what objects the robot is carrying, and you can remove the object.

Using the ARCL payload commands, you can view the number of slots on a robot and see what objects the robot is carrying.

The following commands are supported:

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSet Command (shortcut: ps) on page 194

payloadSlotCount Command (shortcut: psc) on page 196

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPayload Command on page 272

Slots represent containers where the objects (payload) are carried on top of the robot. You can assign a name to the slot numbers that represents the object the robot is to carry from one goal to the next. In the example below, slot 1 is carrying "Books".

```
payloadSet 1 Books
```

To configure the number of slots on a robot, in the custom arguments section on the robot add:

-payloadSlots xx

The default number of slots is 4. Note that slot numbering starts at 1. There is no slot 0; that would indicate there is no payload.

See Also...

Understanding the Commands on page 47
Using ARCL Variables on page 55
Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Creating a Map

ARCL allows you to start and stop creating a map scan, as well as add custom markers to the map while creating the scan. These markers can be sensor readings or other data.

You must enable the ArclScan setting in the ARCL server setup section of the **Configuration > Robot Interface** tab in the MobilePlanner software.

Once enabled, the following commands are supported:

scanAddGoal Command on page 262

scanAddInfo Command on page 264

scanAddTag Command on page 267

scanStart Command on page 269

scanStop Command on page 271

For details on these commands, refer to the ARCL Command Reference on page 70.

To start a map scan using scanStart, provide a name string for the 2d scan file that gets created. No argument is needed for the scanStop command, since only one scan may be active at a time. Also, provide a name and optional description (in that order) with the scanAddGoal command, to place a goal in the map. For more details, see scanStart Command on page 269, scanStop Command on page 271 and scanAddGoal Command on page 262.

The scanAddInfo and scanAddTag commands identify custom objects and then locate markers for them in the map. To use these:

- Define the custom objects with the scanAddInfo command. For details, see scanAddInfo Command on page 264.
- Add markers while scanning at positions throughout the map with the scanAddTag command. For details, see scanAddTag Command on page 267.

See Also...

Understanding the Commands on page 47
Using ARCL Variables on page 55
Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Tracking Sectors

Sectors are designated areas in the map which can trigger certain Motivity tasks, simple messaging or alternative mobile behaviors. For example, a speed sector-enabled robot (set in SetNetGo with the enableSpeedSectors startup parameter) can adopt the defined speed limit for a SlowSector whenever it is located within the bounds of any of that speed sector's type in the map. You can also add your own sectors to the map, with a custom definition, either added manually or using the scanAddInfo command. Then use MobilePlanner to define regions in the map for your sectors.

Enable ARAM tracking for each of the sectors through SetNetGo. Once enabled, ARCL supports several commands that report which sectors your Motivity platform is in based on points, goals and paths:

trackSectors Command on page 279

trackSectorsAtGoal Command on page 282

trackSectorsAtPoint Command on page 285

trackSectorsPath Command on page 288

The response to all commands is a list of the pertinent sectors:

```
TrackSectors: <SectorType> <Sector name, if designated in the map>
...
End of TrackSectors
```

See Also...

Understanding the Commands on page 47
Using ARCL Variables on page 55
Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Navigating and Localizing

The following ARCL commands are available for navigating and localizing the robot.

distanceBetween Command on page 100

distanceFromHere Command on page 102

follow Command on page 116

etaRequest Command on page 111

localizeToPoint Command on page 151

getGoals Command on page 124

goto Command on page 135

gotoPoint Command on page 137

gotoRouteGoal Command on page 139

patrol Command on page 177

patrolOnce Command on page 179

See Also...

Understanding the Commands on page 47
Using ARCL Variables on page 55
Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Using Range Devices and Custom Sensors

ARAM uses range-device readings, both real and virtual, to plan a global path and to detect obstacles along the way so that it can recalculate a local path plan if needed. Range devices include laser- range finders, SONAR, forbidden lines and areas, single-robot sectors, and many others. The following commands are supported in ARCL for using range devices and custom sensors:

customReadingAdd Command on page 96

customReadingAddAbsolute Command on page 94

customReadingsClear Command on page 98

rangeDeviceGetCumulative Command on page 255

rangeDeviceGetCurrent Command on page 257

rangeDeviceList Command on page 259

You can use the ARCL command rangeDeviceList, to retrieve a list from the mobile robot. Like single-robot sectors and forbidden spaces, some ranging sensors are location dependent in that their position in the map is fixed. Other ranging data, like from SONAR, are transient and independent of where the Motivity platform is on the map.

The rangeDeviceGetCurrent command retrieves a series of absolute X and Y map coordinates in millimeters related to the range device's active reflections off an obstacle or relative to the position of the platform (in relation to the virtual sensor). The rangeDeviceGetCumulative command responds with the absolute X and Y map coordinates of persistent readings that ARAM tracks for avoidance while planning a local path.

You can influence local path planning and obstacle avoidance with custom sensors through ARCL. You do this by enabling custom startup arguments in SetNetGo, refer to for details. Your application software then tells ARAM of the custom readings from the device, one at a time. Use customReadingAdd for robot-local coordinates relative to the center of the platform , this is useful for real ranging devices onboard. Use customReadingAbsoluteAdd for map-absolute coordinates, which is useful for location-dependent virtual areas. The customReadingClear command removes all of the particular custom device's ranging data from ARAM if it hadn't already expired.

Enable a custom global planning sensor in order to influence ARAM's global path planner. For example, you can create a custom door global sensor which, at certain times set by your application, is "closed" by customReadAdd data. Once set, ARAM will not plan a global path through the closed door and will treat it as an obstacle while driving past.

See Also...

Understanding the Commands on page 47

Using ARCL Variables on page 55

Using Tasks and Macros on page 56

Using Configuration Commands on page 58

Using the Queuing Commands on page 60

Working With Payloads on page 61

Creating a Map on page 63

Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

Monitoring the I/O Ports

If your hardware and software supports external connections, you can enable or disable the ports for ARCL control in SetNetGo.

Warning! Do not attempt to connect I/O ports if your system did not come with them. If one or more I/O ports are incorrectly assigned or inadvertently triggered, the robot or its systems can be physically damaged. Contact Adept technical support for more information.

You can control and monitor the I/O ports with the following ARCL input and output commands:

analogInputList Command on page 74

analogInputQueryRaw Command on page 75

analogInputQueryVoltage Command on page 76

inputList Command on page 142

inputQuery Command on page 144

outputList Command on page 171

outputOff Command on page 173

outputOn Command on page 174

outputQuery Command on page 175

The following examples show how inputs and outputs can be listed and queried, and how outputs can be turned on/off:

```
inputList
digin1
End of inputList

inputQuery digin1
Input: digin1 off

outputList digout1 digout2
End of outputList

outputQuery digout1
Output: digout1 off

outputOn digout1
Output: digout1 on

outputOff digout1
Output: digout1 off
```

See Also...

Understanding the Commands on page 47

Using ARCL Variables on page 55
Using Tasks and Macros on page 56
Using Configuration Commands on page 58
Using the Queuing Commands on page 60
Working With Payloads on page 61
Creating a Map on page 63
Tracking Sectors on page 64
Navigating and Localizing on page 65
Using Range Devices and Custom Sensors on page 66
Monitoring the I/O Ports on page 68

ARCL Command Reference

This section provides a description of each command in the ARCL command set. The command descriptions are provided in alphabetical order.

See Also...

Introduction to ARCL on page 25
Enable Options in
Set ARCL Parameters in MobilePlanner on page 30
Connect to ARCL Using a Telnet Client on page 42
Using the ARCL Commands on page 46
ARCL Command Reference on page 70
ARCL Server Messages on page 299

analogInputList Command

Lists the named analog inputs.

Syntax

analogInputList

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns a series of "AnalogInputList" in the following format:

```
AnalogInputList: <minV> <maxV> <maxRaw> <name>
```

Details

The analogInputList command returns the list of analog input ports with specs enabled through SetNetGo. <minV> and <maxV> are doubles converted to volts, and <maxRaw> is an integer of the maximum value of the analog to digital conversion (minRaw is always 0); 1023 for a 10-bit A/D converter, for example.

Examples

To view the list of analog input ports, enter the following:

```
analogInputList
```

Related Commands

analogInputQueryRaw Command on page 75

analogInputQueryVoltage Command on page 76

analogInputQueryRaw Command

Queries the state of an analog input by raw.

Syntax

analogInputQueryRaw <name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter the name of the device to query.

Responses

The command returns the state of the specified analog port in the following format:

```
AnalogInputRaw: <name> <rawValue>
```

Details

The analogInputQueryRaw command queries the state of the specified analog input. The data returned by analogInputQueryRaw is an integer called <rawValue>.

To convert the <rawValue> to voltage, use the following equation:

<minVoltage> + (<maxVoltage> - <minVoltage>) * <rawValue> / <maxRaw>

Related Commands

analogInputList Command on page 74

analogInputQueryVoltage Command on page 76

analogInputQueryVoltage Command

Queries the state of an analog input by voltage.

Syntax

analogInputQueryVoltage <name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter the name of the device to query.

Responses

The command returns the state of the specified analog port in the following format:

```
AnalogInputVoltage: <name> <V>
```

where <V> is a double converted to volts.

Details

The analogInputQueryRaw command queries the state of the specified analog input by voltage. The data returned by analogInputQueryVoltage is a voltage as a double between <minVoltage> and <maxVoltage>.

Related Commands

analogInputList Command on page 74

analogInputQueryRaw Command on page 75

applicationFaultClear Command

Clears a named application fault.

Syntax

applicationFaultClear <name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name for the fault.

Responses

The command returns:

```
FaultCleared: Fault_<drivingFault or criticalFault> <name> "<short_desc>" "<long_desc>"
bool_driving bool_critical bool_applicaiton
...
ApplicationFaultClear cleared <name>
```

Details

The faultsGet command returns the list of faults that are currently triggered. For Enterprise Manager, if a robot is unavailable because of a fault, the returned message will start with Fault_ and end with _<name> with the relevant flags in the middle, and each flag will be separated by the underscore character (_).

Examples

The following example clears the application fault named "faulTest2":

```
applicationfaultclear faulTest2
```

The command returns:

```
FaultCleared: Fault_Driving_Critical_Application faulTest2 "Fault test 2" "This is a test
of the driving application fault" true true true
Stopping
ApplicationFaultClear cleared faulTest2
```

Related Commands

applicationFaultQuery Command on page 79

applicationFaultSet Command on page 81

faultsGet Command on page 114

applicationFaultQuery Command

Gets the list of any application faults currently triggered.

Syntax

applicationFaultQuery

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
ApplicationFaultQuery: Fault_<drivingFault or criticalFault> <name> "<short_desc>"
"<long_desc>" bool_driving bool_critical bool_applicaiton
...
End of ApplicationFaultQuery
```

Details

The applicationFaultQuery command returns the list of application faults that are currently triggered. For Enterprise Manager, if a robot is unavailable because of a fault, the returned message will start with Fault_ and end with _<name> with the relevant flags in the middle, and each flag will be separated by the underscore character (_).

This command is related to the faultsGet command. For details, see faultsGet Command on page 114

Examples

The following example shows a listing of the application faults:

```
applicationfaultquery
```

The command returns:

```
ApplicationFaultQuery: Fault_Driving_Critical_Application faulTest2 "Fault test 2" "This
is a test of the driving application fault" true true true
End of ApplicationFaultQuery
```

Related Commands

applicationFaultClear Command on page 77

applicationFaultSet Command on page 81

faultsGet Command on page 114

applicationFaultSet Command

Sets an application fault.

Syntax

applicationFaultSet <name> "<short_description>" "<long_description>" <bool_driving> <bool_critical>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name for the fault.
short_description	Enter a string that will be a brief description of the fault. If the string contains spaces, it must be enclosed in double quotes.
long_description	Enter a string that will be a detailed description of the fault. If the string contains spaces, it must be enclosed in double quotes.
bool_driving	Enter 1 if this is a driving fault; otherwise, enter 0.
bool_critical	Enter 1 if this is a critical fault; otherwise, enter 0.

Responses

The command returns:

```
ApplicationFaultSet set <name>
Fault: Fault_<drivingFault or criticalFault> <name> "<short_desc>" "<long_desc>" bool_
driving bool_critical bool_applicaiton
```

Details

The applicationFaultSet command sets an application fault. All parameters are required. For Enterprise Manager, if a robot is unavailable because of a fault, the returned message will start with Fault_ and end with _<name> with the relevant flags in the middle, and each flag will be separated by the underscore character (_).

Examples

The following example sets a fault named "faulTest":

```
ApplicationFaultSet faulTest "Fault test" "This is a test of the driving application
fault" 1 1
```

The command returns:

```
ApplicationFaultSet set faulTest
Fault: Fault_Driving_Critical_Application faulTest "Fault test" "This is a test of the
driving application fault" true true true
```

Related Commands

applicationFaultClear Command on page 77

applicationFaultQuery Command on page 79

faultsGet Command on page 114

arclSendText Command

Sends the given message to all ARCL clients.

Syntax

arclSendText <string>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

To use this command, you must first enable the -enableTaskArclSendText option in SetNetGo, see Setup Options in for details.

Parameters

The command parameters are described in the following table.

Parameters	Definition
string	Enter a text string that represents the message you want to send to the ARCL clients. Quotes around the string are optional.

Responses

The command returns the following:

```
<string>
```

Details

The arclSendText command sends a message to all ARCL clients. This is an instant task; you can use this command to associate the ArclSendText task with goals and routes.

This is typically used to notify or activate other offboard automation processes in conjunction with the robot's activities. ARAM sends the task's string argument to all ARCL connections.

Example

```
arclsendtext "Entering room, please stand clear."
```

clearAllObstacles Command

Clears all obstacle readings.

CAUTION: DO NOT execute this command while the robot is moving. Otherwise, damage to the robot or other equipment may result.

Syntax

clearAllObstacles

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command, when successful, returns the following response:

```
Cleared all obstacles
```

Details

The clearAllObstacles command clears (removes) all obstacle readings from the robot. Therefore, it *must not* be used while the robot is moving, or the robot could crash and cause damage to itself, its payload or other equipment.

Examples

To clear the robot's obstacle readings, enter:

```
clearAllObstacles
```

The command returns:

```
Cleared all obstacles
```

configAdd Command

Use the configAdd command to enter sections and related configuration parameter keywords and values to the configuration list.

Syntax

configadd <section>

configadd <configuration> <value>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

The command parameters are described in the following table.

Parameters	Definition
section	Enter a text string to represent a name for the new section you want to add to the configuration list.
configuration	Enter a text string to represent a name for the new parameter you want to add to the configuration list.
value	Enter a value for the new parameter.

Responses

The command returns information about the added configuration in the following format:

```
Added  <configuration> <value>
```

Details

When creating the configuration list, you must first identify which section the configuration parameter is associated, and then provide the parameter's keyword and new value. Configuration keywords are case-sensitive.

Examples

```
configAdd Section Files
Added 'Section Files' to the config
```

```
configAdd Map theNewMap.map
Added 'Map theNewMap.map' to the config
```

Related Commands

configParse Command on page 87

configStart Command on page 89

getConfigSectionInfo Command on page 117

getConfigSectionList Command on page 119

getConfigSectionValues Command on page 121

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

configParse Command

Sends the configuration parameters to ARAM, which implements the configuration changes.

Syntax

configParse

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

This command does not have any parameters.

Responses

The command returns information about the added configuration in the following format:

```
Will parse config
Config parsed fine
     -OR-
Config had errors parsing: <errors>
```

Details

The configParse command sends the configuration parameters to ARAM, which implements the configuration changes.

Notice, in the following example, that the "Map changed" response was not generated by ARCL. Rather, it is an ARAM event-warning, which is sent automatically to all attached clients. See Server Messages for details. ARAM catches and reports errors both for configuration and system issues, for example if it is unable to find a file or correctly load a map file.

Examples

```
configParse
Will parse config "Map changed"
Config parsed fine
```

Related Commands

configAdd Command on page 85

configStart Command on page 89

getConfigSectionInfo Command on page 117

getConfigSectionList Command on page 119

getConfigSectionValues Command on page 121

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

configStart Command

Initialize a configuration list, similar to creating a task list. The configStart command overwrites any previous list.

Syntax

configstart

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

This command does not have any parameters.

Responses

The command returns the following information:

```
New config starting
```

Details

Use the configStart command to initialize a configuration list, similar to creating a task list. The configStart command overwrites any previous list.

When creating the configuration list, you must first identify which section the configuration parameter is associated, and then provide the parameter's keyword and new value. Configuration keywords are case-sensitive.

Examples

To start a new configuration, enter the following:

```
configStart
```

The command returns:

```
New config starting
```

Related Commands

configAdd Command on page 85

configParse Command on page 87

getConfigSectionInfo Command on page 117

getConfigSectionList Command on page 119

getConfigSectionValues Command on page 121

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

connectOutgoing Command

Connects (or reconnects) a socket to the specified outside server.

Syntax

connectOutgoing <hostname> <port>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

The command parameters are described in the following table.

Parameters	Definition
hostname	Enter the name of the host (outside server) that you wish to connect to. This can also be entered as the IP address of the host.
port	Enter the port number that will be used for the connection.

Responses

The command returns information about the outgoing connection in the following format:

```
Outgoing connected to <hostname> <port>
```

Details

This command (re)connects a socket to the specified outside server. It is primarily used for debugging purposes.

Examples

To connect to IP 192.168.0.12 with port 5353, enter:

```
connectOutgoing 192.168.0.12 5353
```

To connect to host named "ourhost" with port 5353, enter:

```
connectOutgoing ourhost 5353
```

createInfo Command

Creates a piece of information.

Syntax

createInfo <infoName> <maxLen> <infoValue>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
infoName	Enter a string that will represent the name for the information.
maxLen	Enter the maximum character length that can be used for the information.
infoValue	Enter a string that represents the information value. **NOTE:** If the number of characters in the string exceeds the <maxLen> value, the string will be truncated to that number of characters.

Responses

The command returns information about the new piece of information in the following format:

```
Created info for <infoName>
```

Details

This command is used to create a piece of information that resides on the connected device. Once the information is created, it can be viewed using the getInfo command, or updated using the updateInfo command. For details, see getInfo Command on page 126 and updateInfo Command on page 293.

All information on the connected device can be listed with the getInfoList command. For details, see getInfoList Command on page 128.

Examples

To create a new piece of information called "myString" with a maximum length of 10 characters and an initial value of "testing", enter the following:

```
createinfo myString 10 testing
```

The command returns:

```
Created info for myString
```

Related Commands

getInfo Command on page 126

getInfoList Command on page 128

updateInfo Command on page 293

customReadingAddAbsolute Command

Adds a sensor reading in absolute (map) coordinates.

Syntax

customReadingAddAbsolute<name> <X> <Y>

Usage Considerations

This ARCL command is only available on the robot.

There is no space between the command and the sensor name. See the examples section.

This parameter is case-sensitive.

This adds the device in absolute (map) coordinates. To add the device relative to the robot, use the customReadingAdd command. For details, see customReadingAdd Command on page 96.

ARAM Settings

This command requires the addition of the "-customSensor <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name for the device. This parameter is case-sensitive.
X	Enter the X map coordinate (in mm).
Y	Enter the Y map coordinate (in mm).

Responses

The command returns:

```
Added absolute reading <X> <Y>
```

Details

The customReadingAddAbsolute command adds a sensor reading that is at <X> <Y> millimeters in absolute (map) coordinates. For example, an entry of 200 100 would be a point that is 200 mm in front of the robot and 100 mm to the left of the robot.

This parameter is case-sensitive.

Note that this command adds the device in absolute (map) coordinates. To add the sensor reading in robot-relative coordinates, use the customReadingAdd command. For details, see customReadingAdd Command on page 96.

Examples

> **NOTE:** The following example assumes a custom sensor named "ARCL_CustomSensor" was previously added to the system. For details, see the ARAM Settings section.

To add a sensor reading at absolute (map) coordinates X: -2532 Y: 5471, enter the following:

```
customReadingAddAbsoluteARCL_CustomSensor -2532 5471
```

The command returns:

```
Added absolute reading -2532 5471
```

Related Commands

customReadingAdd Command on page 96

customReadingAddAbsolute Command on page 94

customReadingsClear Command on page 98

rangeDeviceGetCumulative Command on page 255

rangeDeviceGetCurrent Command on page 257

rangeDeviceList Command on page 259

customReadingAdd Command

Adds a sensor reading in robot-relative coordinates.

Syntax

customReadingAdd<name> <X> <Y>

Usage Considerations

This ARCL command is only available on the robot.

There is no space between the command and the sensor name. See the examples section.

This parameter is case-sensitive.

This adds the sensor reading in relative (to the robot base) coordinates. To add the device in absolute (map) coordinates, use the customReadingAddAbsolute command. For details, see customReadingAddAbsolute Command on page 94.

ARAM Settings

This command requires the addition of the "-customSensor <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name for the device. This parameter is case-sensitive.
X	Enter the X coordinate (in mm) relative to the robot base.
Y	Enter the Y coordinate (in mm) relative to the robot base.

Responses

The command returns:

```
Added reading <X> <Y>
```

Details

The customReadingAdd command adds a sensor reading that is at <X> <Y> millimeters in robot-relative coordinates (where +X is in front of the robot, +Y is to the left of the robot). For example, an entry of 200 100 would be a point that is 200 mm in front of the robot and 100 mm to the left of the robot.

This parameter is case-sensitive.

Note that this command adds the device in robot-relative (to the robot base) coordinates. To add the sensor reading in absolute (map) coordinates, use the customReadingAddAbsolute command. For details, see customReadingAddAbsolute Command on page 94.

Examples

> **NOTE:** The following example assumes a custom sensor named "ARCL_CustomSensor" was previously added to the system. For details, see the ARAM Settings section.

To add a sensor reading at robot-relative coordinates X: 100 Y: 0, enter the following:

```
customReadingAddARCL_CustomSensor 100 0
```

The command returns:

```
Added reading 100 0
```

Related Commands

customReadingAdd Command on page 96

customReadingAddAbsolute Command on page 94

customReadingsClear Command on page 98

rangeDeviceGetCumulative Command on page 255

rangeDeviceGetCurrent Command on page 257

rangeDeviceList Command on page 259

customReadingsClear Command

Clears all the named sensor readings from ARAM.

Syntax

customReadingsClear<name>

Usage Considerations

This ARCL command is only available on the robot.

There is no space between the command and the sensor name. See the examples section.

ARAM Settings

This command requires the addition of the "-customSensor <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name for the device.

Responses

The command returns:

```
Cleared readings
```

Details

The customReadingClear command clears all custom senor readings that were added using the customReadingAdd command or the customReadingAddAbsolute command. For details on these commands, use the links in the Related Commands section.

Examples

To clear the custom sensor readings, enter the following:

```
customReadingsClearARCL_CustomSensor
```

The command returns:

```
Cleared readings
```

Related Commands

customReadingAdd Command on page 96

customReadingAddAbsolute Command on page 94

customReadingsClear Command on page 98

rangeDeviceGetCumulative Command on page 255

rangeDeviceGetCurrent Command on page 257

rangeDeviceList Command on page 259

distanceBetween Command

Finds the path distance between two given goals.

Syntax

distancebetween <FromGoal> <ToGoal>

Usage Considerations

This ARCL command is only available on the robot.

This command should only be used when the robot is idle and stationary.

Parameters

The command parameters are described in the following table.

Parameter	Definition
FromGoal	Enter the name of the first goal.
ToGoal	Enter the name of the second goal.

Responses

The command returns:

```
Will find distance between "<FromGoal>" and "<ToGoal>"
DistanceBetween: <mm> "<FromGoal>" "<ToGoal>"
```

Details

The distanceBetween command plans a path from goal to goal and reports that path distance. It assumes, of course, there are no unforeseen obstacles. This command is processing-intensive. Therefore, it should be used only when the robot is idle and stationary. Additionally, be sure to update the distance after any changes are made to the map.

To find the distance from the current robot position to a specified goal, use the distanceFromHere command. For details, see distanceFromHere Command on page 102.

Examples

The following example finds the distance between goals "g_6" and "g_7".

```
distancebetween g_6 g_7
```

The command returns:

```
Will find distance between "g_6" and "g_7"
DistanceBetween: 13183 "g_6" "g_7"
```

Related Commandss

distanceFromHere Command on page 102

distanceFromHere Command

Finds the path distance from the current robot position to a given goal.

Syntax

distanceFromHere <ToGoal>

Usage Considerations

This ARCL command is only available on the robot.

This command should only be used when the robot is idle and stationary.

Parameters

The command parameters are described in the following table.

Parameter	Definition
ToGoal	Enter the name of the goal.

Responses

The command returns:

```
Will find distance from here to <ToGoal>.
DistanceBetween: <mm> "<to goal>"
```

Details

The distanceFromHere command plans a path from the current robot position to the specified goal and reports that path distance. It assumes, of course, there are no unforeseen obstacles. This command is processing-intensive. Therefore, it should be used only when the robot is idle and stationary. Additionally, be sure to update the distance after any changes are made to the map.

To find the distance between two specified goals, use the distanceBetween command. For details, see distanceBetween Command on page 100.

Examples

The following example finds the distance from the current robot position to goal "g_5".

```
distancefromhere g_5
```

The command returns:

```
Will find distance from here to "g_5"
DistanceFromHere: 9960 "g_5"
```

Related Commands

distanceBetween Command on page 100

dock Command

Sends the robot to the dock.

Syntax

dock

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
DockingState: <dock_state> ForcedState: <forced_state> ChargeState: <charge_state>
```

Details

The dock command sends the robot to the dock so it can recharge.

When the robot is fully-charged, it will automatically undock from the dock/recharge station.

You can undock the robot with the undock command, or by using one of the "goto..." commands. For details on these commands, use the links in the Related Commands section.

Examples

The following example docks the robot:

```
dock
```

The command returns:

```
DockingState: Undocked ForcedState: Unforced ChargeState: Unknowable
DockingState: Docking ForcedState: Unforced ChargeState: Unknowable
```

Related Commands

goto Command on page 135

gotoPoint Command on page 137

gotoRouteGoal Command on page 139

undock Command on page 291

doTask Command

Performs a single task.

Syntax

doTask <task> <argument>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
task	Enter the name of the task you want the mobile robot to perform, such as a goto task.
argument	Enter the appropriate arguments for the task you want the robot to perform. Using the goto task example, you would need to enter a goal name, such as goto goal_1. Enclose any string arguments in double quotes.

Responses

The command returns:

```
Will do task <task> <argument>
Doing task <task> <argument>
...
Completed doing task <task> <argument>
```

Details

The doTask command tells the robot to perform a single task. The task is carried out immediately. This task is similar to the doTaskInstant command, which performs "instant tasks" immediately. For details, see doTaskInstant Command on page 107.

Examples

The following example tells the robot to go to goal g_5:

```
dotask goto g_5
```

The command returns:

```
Will do task goto g_5
```

```
Doing task goto g_5
Completed doing task goto g_5
```

The following example tells the robot to wait for 10 seconds:

```
doTask wait 10
```

The command returns:

```
Will do task wait 10
Doing task wait 10
WaitState: Waiting 10 seconds with status "Waiting"
WaitState: Waiting completed
Completed doing task wait 10
```

Related Commands

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

doTaskInstant Command

Performs an instant task.

Syntax

doTaskInstant <task> <argument>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
task	Enter the name of the instant task you want the mobile robot to perform.
argument	Enter the appropriate arguments for the instant task you want the robot to perform. Enclose strings in double quotes.

Responses

The command returns:

```
Completed doing instant task <task> <argument>
```

Details

The doTaskInstant command tells the mobile robot to immediately perform the specified task. You can only use "instant tasks" with the doTaskInstant command. This command is similar to the doTask command. For details, see doTask Command on page 105.

The following are examples of two instant tasks that are available for use with ARCL.

- movementParametersTemp - Sets the movement parameters temporarily (this route and/or this mode).
- pathPlanningSettingsTemp - Sets the path-planning parameters temporarily (this route and/or this mode).

The list of available instant tasks can be viewed using the MobilePlanner software. For details, see the *Adept Motivity User's Guide*.

Related Commands

doTask Command on page 105

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

echo Command

Enables/disables echo, or returns the current echo state.

Syntax

echo [state]

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

The command parameters are described in the following table.

Parameter	Definition
state	Optional. Enter "on" to enable echo; enter "off" to disable echo. If omitted, the command returns the current echo state.

Responses

The command returns:

```
Echo is <state>
```

-Or-

```
Echo turned <state>
```

Examples

The following command returns the current echo state:

```
echo
Echo is off.
```

The following command turns echo on:

```
echo on
Echo turned on.
```

The following command turns echo off:

```
echo off
Echo turned off.
```

enableMotors Command

Enables the robot motors, if the robot was not E-stopped.

Syntax

enableMotors

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Motors are enabled
```

However, if an E-stop was pressed on the robot, the following message is displayed.

```
Estop pressed cannot enable motors
```

Examples

The following command enables the robot motors:

```
enablemotors
```

The command returns:

```
Motors are enabled
```

Related Commands

queryMotors Command on page 207

etaRequest Command

Syntax

etaRequest

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
eta <seconds> <distance_mm>
```

Details

The etaRequest command returns the estimated time (in seconds) and distance (in mm) to reach the goal. If the robot is not traveling to a goal, the command returns 0 for both values.

Examples

To get the estimated time (and distance) before the robot reaches the goal, enter:

```
etarequest
```

The command returns:

```
eta 17 25449
```

Related Commands

status Command on page 276

executeMacro Command

Executes the specified macro.

Syntax

executeMacro <macro_name >

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
macro_name	Enter the name of the macro you want the mobile robot to perform.

Responses

The command returns:

```
Executing macro <macro_name>
WaitState: <wait_status>
...
Completed macro <macro_name>
```

Details

Use this command to execute a specified macro found on the map. You can use the MobilePlanner software to create macros. For details, see the *Adept Motivity User's Guide*.

Use the getMacros command to display a list of the macros available in ARCL. For details, see getMacros Command on page 130.

Example

The following example executes the macro named "Adept Greeting".

```
executemacro Adept Greeting
```

The command returns:

```
Executing macro Adept Greeting
WaitState: Waiting 1 seconds with status "Waiting"
WaitState: Waiting completed
Completed macro Adept Greeting
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

faultsGet Command

Gets the list of any faults currently triggered.

Syntax

faultsGet

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
FaultList: Fault_<drivingFault or criticalFault> <name> "<short_desc>" "<long_desc>"
bool_driving bool_critical bool_applicaiton
...
End of FaultList
```

For Enterprise Manager, if a robot is unavailable because of a fault, the returned message will start with Fault_ and end with _<name> with the relevant flags in the middle, and each flag will be separated by the underscore character (_).

Details

The faultsGet command returns the list of faults that are currently triggered—this includes system-generatd faults and application-generated faults. Application faults can be set using the applicationFaultSet command, cleared using the applicationFaultClear command, and queried using the applicationFaultQuery command. For details on these commands, see the related commands section.

Examples

The following example shows a listing of the faults:

```
faultsget
```

The command returns:

```
FaultList: Fault_Driving_Application faultTest "Fault test" "This is a test of the
application fault" true false true
FaultList: Fault_Critical_Application faulTest2 "Fault test 2" "This is a test of the
application fault two" false true true
End of FaultList
```

Related Commands

applicationFaultClear Command on page 77

applicationFaultQuery Command on page 79

applicationFaultSet Command on page 81

log Command on page 153

follow Command

Follow a person walking ahead of the robot.

Syntax

follow

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Following
```

Details

The follow command engages the leg-following behavior for no-hands operation. When this command is executed, the robot will follow (at a safe distance) behind a person who is walking ahead of it.

Examples

To engage the leg-following behavior, enter the following:

```
follow
```

The command returns:

```
Following
```

Related Commands

goto Command on page 135

gotoPoint Command on page 137

gotoRouteGoal Command on page 139

patrol Command on page 177

patrolOnce Command on page 179

getConfigSectionInfo Command

Displays details about the configuration parameters in a specified section.

Syntax

getConfigSectionInfo <section>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

The getConfigSectionInfo arguments are described in the table below.

Parameters	Definition
section	Enter the name of the section from which you want to see a list of configuration parameters. This is a text string; it is case-sensitive. The string must not be enclosed in double quotes.

Responses

When using the getConfigSectionInfo command, ARCL displays the following information:

```
getconfigsectioninfo "<type>" "<type>"
```

Then for each parameter in the section, ARCL displays the following information:

```
GetConfigSectionInfo: <type> <name> <priority> <min> <max> "<description>" "<display
hint>"
...
EndOfGetConfigSectionInfo
```

Details

The getConfigSectionInfo command displays details about the configuration parameters in a specified section. See Examples for details.

Note that a valid section name must be entered, and the section name is case-sensitive.

Use the getConfigSectionList Command to display a list of available sections. For details, see getConfigSectionList Command on page 119.

Examples

The following example displays details about the configuration parameters in the section"Outgoing ARCL Commands".

```
getconfigsectioninfo Outgoing ARCL Commands
GetConfigSectionInfo: "" "SEPARATE_SECTION"
GetConfigSectionParamInfo: Separator
GetConfigSectionParamInfo: Bool LogOutgoingCommands Advanced None None "True
to log outgoing commands from below, false not to" "(null)"
GetConfigSectionParamInfo: Separator
GetConfigSectionParamInfo: String OutgoingCommands1 Advanced None None "ARCL
command(s) to call on the outgoing socket" "(null)"
GetConfigSectionParamInfo: Double OutgoingCommands1Seconds Advanced 0 inf "Call
the command every this many seconds (note it's a double so you can do .5 for half a
second)
0 disables (seconds)" "(null)"
```

Related Commands

configAdd Command on page 85

configParse Command on page 87

configStart Command on page 89

getConfigSectionList Command on page 119

getConfigSectionValues Command on page 121

configAdd Command on page 85

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

getConfigSectionList Command

Displays the list of sections enabled in the ARAM configuration parameters.

Syntax

getConfigSectionList

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

This command does not have any parameters.

Value

Details

The getConfigSectionList displays the list of sections enabled in the ARAM configuration parameters. See Examples for a sample list.

This command would be used in conjunction with getConfigSectionInfo, which returns information about a specified section. For details, see getConfigSectionInfo Command on page 117.

Examples

The following example returns a list of sections that are enabled in the ARAM configuration parameters.

```
getconfigsectionlist
GetConfigSectionList: Log Config
GetConfigSectionList: Connection timeouts
GetConfigSectionList: ARCL server setup
GetConfigSectionList: Outgoing ARCL connection setup
GetConfigSectionList: Outgoing ARCL commands
GetConfigSectionList: Files
GetConfigSectionList: Path Planning Settings
GetConfigSectionList: Debug log
GetConfigSectionList: lms2xx_1 Settings
GetConfigSectionList: Localization settings
GetConfigSectionList: Instant Macro Button Settings
GetConfigSectionList: Periodic Macros
GetConfigSectionList: Driving problem response
```

```
GetConfigSectionList: bumpers Settings
GetConfigSectionList: Teleop settings
GetConfigSectionList: Robot config
GetConfigSectionList: Destination Drawing
GetConfigSectionList: Patrol
GetConfigSectionList: Docking
GetConfigSectionList: Move settings
GetConfigSectionList: Follow (laser) settings
GetConfigSectionList: A/V Config
GetConfigSectionList: Speech Synthesis
GetConfigSectionList: MultiRobot
GetConfigSectionList: Data Log Settings
EndOfGetConfigSectionList
```

Related Commands

configAdd Command on page 85

configParse Command on page 87

configStart Command on page 89

getConfigSectionInfo Command on page 117

getConfigSectionValues Command on page 121

configAdd Command on page 85

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

getConfigSectionValues Command

Displays the current parameter values for the specified section.

Syntax

getConfigSectionValues <section>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

The command parameters are described in the following table.

Parameters	Definition
section	Enter the name of the section from which you want to see a list of parameter values. This is a text string; it is case-sensitive. The string must not be enclosed in double quotes.

Responses

The command returns:

```
GetConfigSectionValue: <value>
...
EndOfGetConfigSectionValues
```

Details

The getConfigSectionValues command displays a list of the specified section's current parameter values. See Examples for a sample listing.

It is typically used with the getConfigSectionList command, which lists the available sections, and the getConfigSectionInfo command, which displays the information for a specified section.

Examples

The following example displays the parameter values for the section "Outgoing ARCL Commands".

```
getconfigsectionvalues Outgoing ARCL Commands
GetConfigSectionValue: LogOutgoingCommands true
```

```
GetConfigSectionValue: OutgoingCommands1
GetConfigSectionValue: OutgoingCommands1Seconds 0
GetConfigSectionValue: OutgoingCommands2
GetConfigSectionValue: OutgoingCommands2Seconds 0
GetConfigSectionValue: OutgoingCommands3
GetConfigSectionValue: OutgoingCommands3Seconds 0
GetConfigSectionValue: OutgoingCommands4
GetConfigSectionValue: OutgoingCommands4Seconds 0
GetConfigSectionValue: OutgoingCommands5
GetConfigSectionValue: OutgoingCommands5Seconds 0
EndOfGetConfigSectionValues
```

Related Commands

configAdd Command on page 85

configParse Command on page 87

configStart Command on page 89

getConfigSectionInfo Command on page 117

getConfigSectionList Command on page 119

configAdd Command on page 85

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

getDateTime Command

Returns the system date and time.

Syntax

getDateTime

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Parameters

This command does not have any parameters.

Examples

To view the current system date and time, enter:

```
getdatetime
```

The command returns:

```
DateTime: 05/03/2012 04:48:55
```

getGoals Command

Returns a list of goal names found in the current map.

Syntax

getGoals

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Goal: <name>
...
Goal: <name>
End of goals
```

Examples

To get a list of the goal names in the current map, enter the following:

```
getgoals
```

The command returns:

```
Goal: w200
Goal: Y
Goal: X
Goal: First_Goal
Goal: goal space
Goal: T
Goal: w180
Goal: First
Goal: V
Goal: w20
Goal: z
End of goals
```

Related Commands

getGoals Command on page 124

getRoutes Command on page 134

goto Command on page 135

gotoRouteGoal Command on page 139

getInfo Command

Returns the string associated with the information name.

Syntax

getInfo <infoName>

Usage Considerations

This ARCL command is only available on the robot.

You can view the value of any information on the connected device—it is not restricted to the information created with the createInfo command. For details, see createInfo Command on page 92.

Parameters

The command parameters are described in the following table.

Parameter	Definition
<infoName>	Enter the name of the information you want to view.

Responses

The command returns:

```
Info: <label> <string_value>
```

Details

The getInfo command returns the information associated with the specified information name. You can use the command to view the value of any information on the connected device. To see a list of all information names on the device, use the getInfoList command. For details, see getInfoList Command on page 128.

Examples

To view the information associated with the information name "mystring", enter the following:

```
getinfo mystring
```

The command returns:

```
Info: Flags 400
```

Related Commands

createInfo Command on page 92

getInfoList Command on page 128

updateInfo Command on page 293

getInfoList Command

Returns the list of information names.

Syntax

getInfoList

Usage Considerations

This ARCL command is only available on the robot.

This command lists all information names on the connected device—it is not restricted to the names created with the createInfo command. For details, see createInfo Command on page 92.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
InfoList: <info>
...
InfoList: <info>
End of info list
```

Details

The getInfoList command is used to list all the information names on the connected device. The list includes the system information names and any user-created information names that were added with the createInfo command. For details, see createInfo Command on page 92.

Examples

To view the list of information names, enter the following:

```
getinfolist
```

The command returns:

```
InfoList: Odometer(KM)
InfoList: LaserUncertainty
InfoList: LaserScore
InfoList: LaserLock
InfoList: LaserNumSamples
InfoList: Flags
InfoList: Fault flags
InfoList: MPacs
InfoList: lms2xx_1 Pacs
InfoList: CPU Use
InfoList: SBC Uptime
```

```
InfoList: ARAM Uptime
InfoList: Idle
InfoList: Queue ID
InfoList: Queue Job ID
InfoList: DebugLogState
InfoList: DebugLogSeconds
InfoList: mystring
End of info list
```

Related Commands

createInfo Command on page 92

getInfo Command on page 126

updateInfo Command on page 293

getMacros Command

Displays a list of macros found in the current map.

Syntax

getmacros

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
<macro_name>
...
<macro_name>
End of macros
```

Details

The getMacros command provides a list of the macro names found in the current map.

Use this command with the executeMacro command. For details, see executeMacro Command on page 112.

Examples

```
getmacros
```

The command returns:

```
Macro_1
Macro_2
Macro_3
End of macros
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

getPayload Command

Gets the payload name.

Syntax

getPayload

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
payload <payload>
```

Details

The getPayload command gets the name of the robot payload. To set the payload name, use the setPay-load command. For details, see setPayload Command on page 272.

Examples

The following example requests the payload name:

```
getpayload
```

The command returns:

```
payload This has widgets
```

Related Commands

payloadQuery Command (shortcut: pq) on page 187

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSet Command (shortcut: ps) on page 194

payloadSlotCount Command (shortcut: psc) on page 196

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPayload Command on page 272

getPrecedence Command

Displays the precedence information for the robot, which is used in a multi-robot encounter. Lower values take higher precedence.

Syntax

getprecedence

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
getPrecedence: <value>
```

Details

The getPrecedence command is used to display the precedence information for the robot. The precedence value is used in a multi-robot encounter. The robot that has the lowest value will get highest precedence, the robot with the next lowest value will get the next highest precedence, and so on. The precedence value is set using the setPrecedence command. For details, see setPrecedence Command on page 274.

Examples

To get the precedence information for the robot, enter the following:

```
getprecedence
```

The command returns:

```
getPrecedence: 0
```

Related Commands

setPrecedence Command on page 274

getRoutes Command

Displays the list of route names found on the current map.

Syntax

getRoutes

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Routes
Route: <routeName>
...
Route: <routeName>
End of routes
```

Examples

To show the list of routes on the current map, enter the following:

```
getroutes
```

The command returns:

```
Routes
Route: tv
Route: xyz
Route: yzx
Route: zy
End of routes
```

Related Commands

getGoals Command on page 124

gotoRouteGoal Command on page 139

goto Command

Sends the robot to the named goal and, when it arrives, turns the robot to the specified heading, if specified.

Syntax

goto <goal_name> [heading]

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The goto arguments are described in the table below.

Parameter	Definition
goal_name	Enter the name of the goal you want the robot to drive to.
heading	Enter an optional heading in degrees.

Responses

The command returns:

```
Going to <goal_name> with heading [heading]
Arrived at <goal_name> with heading [heading]
```

Details

The goto command sends the robot to the named goal and, when it arrives, turns the robot to the specified heading if it was specified.

Examples

To have the robot go to goal "g_15", enter the following:

```
goto g_15
```

The command returns:

```
Going to g_15
Arrived at g_15
```

To have the robot go to goal "X" with a heading of "10", enter the following:

```
goto X 10
```

The command returns:

```
Going to X with heading 10
Arrived at X with heading 10
```

Related Commands

gotoPoint Command on page 137

gotoRouteGoal Command on page 139

gotoPoint Command

Sends robot to the specified point (optional heading, on arrival).

Syntax

gotoPoint <X> <Y> <heading: optional>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
X	Specifies the distance of travel in the robot X direction.
Y	Specifies the distance of travel in the robot Y direction.
heading: optional	Specifies an optional heading in degrees.

Responses

The command returns:

```
Going to point <x> <y> <heading:optional>
Arrived at point <x> <y> <heading:optional>
```

Details

The gotoPoint command sends the robot to the specified point in the map and, upon arrival, turn to the given heading, if it was specified.

Examples

The following is a gotoPoint command with a heading:

```
gotopoint 14000 25000 180
Going to point 14000 25000 180
Arrived at point 14000 25000 180
```

The following is a gotoPoint command without a heading:

```
gotopoint 13000 26000
Going to point 13000 26000
Arrived at point 13000 26000
```

Related Commands

goto Command on page 135

gotoRouteGoal Command on page 139

gotoRouteGoal Command

Sends robot to the specified goal within the specified route (optional route index can be asserted).

Syntax

gotoRouteGoal <route_name> <goal_name> [index]

Usage Considerations

This ARCL command is only available on the robot.

The optional index parameter is typically reserved for specific applications and is not required under normal circumstances.

Parameters

The command parameters are described in the following table.

Parameter	Definition
route_name	Enter the name of the route where the robot will find the goal.
goal_name	Enter the name of the goal you want the robot to navigate to.
index	Enter an optional index. This is typically reserved for specific applic-ations and is not required under normal circumstances.

Responses

The command returns:

```
Going to <goal_name>
Arrived at <goal_name>
```

Details

Sends robot to specified goal within the specified goal. Optionally, the command also asserts a route index if one has been specified. However, this is typically reserved for specific applications and is not required under normal circumstances.

Examples

To send the robot to goal "g_17" in the route named "test", enter the following:

```
gotoroutegoal test g_17
```

The command returns:

```
Going to g_17
Arrived at g_17
```

Related Commands

goto Command on page 135

gotoPoint Command on page 137

help Command

Provides a list and brief description of available ARCL commands.

Syntax

help

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

This command does not have any parameters.

Details

The help command provides a list and brief description of the available ARCL commands on the connected server or robot. The list shown depends on the current configuration of your server or robot, therefore, it may not show the entire library of commands.

Examples

To view the command list and descriptions, enter the following:

```
help
```

The command returns:

> **NOTE:** The list of available commands depends on your system configuration.

```
Commands:
      addCustomCommand    Adds a custom command that sends a message out ARCL when
                         called
addCustomStringCommand    Adds a custom string command that sends a message out ARCL when
                         called
          arclSendText    Sends the given message to all ARCL clients
       connectOutgoing    (re)connects a socket to the given outside server
                  echo    with no args gets echo, with args sets echo
                …
        queueShowRobot    shows the status of all the robots [qsr]
                  quit    closes this connection to the server
End of commands
```

inputList Command

Lists the named digital inputs.

Syntax

inputList

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Input: <name>
...
End of InputList
```

Details

The inputList command returns the list of digital inputs. To get the status of a particular digital input, use the inputQuery command. For details, see inputQuery Command on page 144.

Examples

To get the list of digital inputs, enter the following:

```
inputlist
```

The command returns:

```
InputList: out_one
InputList: out_two
End of InputList
```

Related Commands

inputQuery Command on page 144

outputList Command on page 171

outputOff Command on page 173

outputOn Command on page 174

outputQuery Command on page 175

inputQuery Command

Queries the state of a named input.

Syntax

inputQuery <name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter the name of the input to query.

Responses

The command returns:

```
Input: <name> <status>
```

Details

The outputQuery command returns the status of the named digital input. To get a list of the digital inputs, use the inputList command. For details, see inputList Command on page 142.

Examples

To get the status of digital input named "in_one", enter the following:

```
Inputquery in_one
```

The command returns:

```
Input: in_one off
```

Related Commands

inputList Command on page 142

outputList Command on page 171

outputOff Command on page 173

outputOn Command on page 174

outputQuery Command on page 175

listAdd Command

Adds a task to the task list.

Syntax

listAdd <task> <argument>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The listAdd arguments are described in the table below.

Parameters	Definition
task	Enter the name of the task you want to add to the task list, such as a goto task.
argument	Enter the appropriate arguments for the task you want the robot to perform. Using the goto task example, you would need to enter a goal name, such as goto goal_1. Strings must be enclosed in double quotes.

Responses

The command returns:

```
Added task <task> <argument>
...
```

Details

ARCL allows you to create a task list, add tasks to the list, and then execute the task list. In doing so, you can make use of the tasks that are available in MobilePlanner for building routes and macros.

You must first initialize a task list with the listStart command. This overwrites any list that has already started but has not been executed. For details, see listStart Command on page 149.

Use the listAdd command with a task name as an argument, followed by any other task argument. This puts the task into the current list.

When the list is complete, use listExecute to perform the series of tasks in the order they were entered. The list can be executed only once. For details, see listExecute Command on page 147.

Examples

As shown below, the list commands allow you to assemble and execute a sequence of tasks. In the following example, the robot travels to the goal *Lobby* and when it gets there, says its name and then asks for a name:

```
liststart
List being cleared
Making new list

listadd goto Lobby
Added task goto Lobby
listadd say "My name is PatrolBot"
Added task say "My name is PatrolBot"
listadd say "What is your name?"
Added task say "What is your name?"

listexecute
Executing list
Successfully finished task list
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

listExecute Command

Use listExecute to perform the series of tasks, with the order of first in, first out; the list can be executed only once.

Syntax

listExecute

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Executing list
Successfully finished task list
```

Details

ARCL allows you to create a task list, add tasks to the list, and then execute the task list. In doing so, you can make use of the tasks that are available in MobilePlanner for building routes and macros.

You must first initialize a task list with the listStart command. This overwrites any list that has already started but has not been executed. For details, see listStart Command on page 149.

Use the listAdd command with a task name as an argument, followed by any other task argument. This puts the task into the current list. For details, see listAdd Command on page 145.

When the list is complete, use listExecute to perform the series of tasks in the order they were entered. The list can be executed only once.

Examples

As shown below, the list commands allow you to assemble and execute a sequence of tasks. In the following example, the robot travels to the goal *Lobby* and when it gets there, says its name and then asks for a name:

```
liststart
List being cleared
Making new list

listadd goto Lobby
Added task goto Lobby
listadd say "My name is PatrolBot"
Added task say "My name is PatrolBot"
listadd say "What is your name?"
```

```
Added task say "What is your name?"

listexecute
Executing list
Successfully finished task list
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

listStart Command

Initializes a new task list for the robot to perform.

Syntax

listStart

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
List being cleared
Making new list
```

Details

ARCL allows you to create a task list, add tasks to the list, and then execute the task list. In doing so, you can make use of the tasks that are available in MobilePlanner for building routes and macros.

You must first initialize a task list with the listStart command. This overwrites any list that has already started but has not been executed.

Use the listAdd command with a task name as an argument, followed by any other task argument. This puts the task into the current list. For details, see listAdd Command on page 145.

When the list is complete, use listExecute to perform the series of tasks in the order they were entered. The list can be executed only once. For details, see listExecute Command on page 147.

Examples

As shown below, the list commands allow you to assemble and execute a sequence of tasks. In the following example, the robot travels to the goal *Lobby* and when it gets there, says its name and then asks for a name:

```
liststart
List being cleared
Making new list

listadd goto Lobby
Added task goto Lobby
listadd say "My name is PatrolBot"
Added task say "My name is PatrolBot"
listadd say "What is your name?"
```

```
Added task say "What is your name?"

listexecute
Executing list
Successfully finished task list
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

localizeToPoint Command

Localizes to a given point, optionally with spread.

Syntax

localizeToPoint <X> <Y> <T> [xySpread] [thSpread]

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
X	Enter an integer (in mm) that represents the X coordinate where you want the robot to localize.
Y	Enter an integer (in mm) that represents the Y coordinate where you want the robot to localize.
T	Enter an integer that represents the Theta value in degrees.
xySpread	Enter an optional integer (mm) that represents the XY spread.
thSpread	Enter an optional integer (degrees) that represents the Theta spread.

Responses

The command returns:

```
Localized to point
```

Details

Localizes to a given point, optionally with XY and Theta spread.

Examples

The following example localizes the robot to point XYT point 100 100 0:

```
localizetopoint 100 100 0
Localized to point
```

The following example localizes the robot to the previous XYT point, but it includes an XY spread of 10 mm and a Theta spread of 2 degrees:

```
localizetopoint 100 100 0 10 2
Localized to point
```

Related Commands

gotoPoint Command on page 137

log Command

Logs the message to the normal log file.

Syntax

log <message> [level]

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

The command parameters are described in the following table.

Parameter	Definition
message	Enter the string the will be the log message. If it contains any spaces, the string must be enclosed in double quotes.
level	Enter the optional level for the message: Terse, Normal, or Verbose: Terse = Used for critical errors Normal = Used for standard error information Verbose = Used for routine information that typically doesn't need to be seen. If no level is specified, it defaults to the "Normal" level.

Responses

The command returns:

```
Logging '<message>' with level [level]
```

Details

The log command is used to add user-created messages to the system log file. There are three levels that can be optionally specified for the message; if none is specified, the default level of "Normal" is used.

Examples

The following example logs the message "This is a test" with no level specified:

```
log "This is a test" terse
Logging 'This is a test' with level Normal
```

The following example logs the message "This is a test" with a level of "Terse":

```
log "This is a test" terse
Logging 'This is a test' with level Terse
```

Related Commands

faultsGet Command on page 114

mapObjectInfo Command

Gets the information about a named map object.

Syntax

mapObjectInfo <name>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name of the map object.

Responses

The command returns:

```
MapObjectInfo: "<name>" <type> "<description>"
MapObjectInfoParams: "<name>" <params>
End of MapObjectInfo
```

Note that if there are no parameters the MapObjectInfoParams will not be shown. The <params> will show all the parameters that are present; strings will be in quotes (if they contain spaces)—those should be looked for and removed. The <description> is what the user enters in the MobilePlanner software.

Details

The mapObjectInfo command displays information about a specific map object. See Examples for details.

There are four related commands that are used to get information about map objects: mapObjectTypeList, mapObjectTypeInfo, mapObjectList, and mapObjectInfo. These can be used in one of two ways:

- Exploratory - by getting broad/general information and "drilling down" to the desired specific information. For this method, you would:
  - Use mapObjectTypeList to show the map object <type>s.
  - Use mapObjectTypeInfo <type> to see if it has parameters or other information.
  - Use mapObjectList <type> to get the <name> of the map objects of that type.
  - Use mapObjectInfo <name> to get information about each map object (this is mostly for those that have parameters).
- Direct - by going after information on a specific map object. For this method, you would:

- Use mapObjectInfo <name> to find out its <type> and its parameters.
- Use mapObjectTypeInfo <type> to see what parameters it has and what they mean. This step isn't needed if you already know what the parameters mean. However, it can be useful for verifying ordering and other details.

For more details on these commands, see the links in the Related Commands section.

Examples

The following example returns information about the map object named "PreferredDirectionRightSingle1":

```
mapobjectinfo PreferredDirectionRightSingle1
```

The command returns:

```
MapObjectInfo: "PreferredDirectionRightSingle1" DriveOnRightSector ""
MapObjectInfoParams: "PreferredDirectionRightSingle1" true 300
End of MapObjectInfo
```

Related Commands

mapObjectList Command on page 157

mapObjectTypeInfo Command on page 159

mapObjectTypeList Command on page 161

mapObjectList Command

Gets the names of map objects of a given type.

Syntax

mapObjectList <type>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

The command parameters are described in the following table.

Parameters	Definition
type	Enter a string that represents the type of map objects you want to list. The string must not contain spaces. The string must not be enclosed in double quotes.

This is a text string; it is case-sensitive.

Responses

The command returns:

```
MapObjectList: "<name>" <type>
MapObjectList: "<name>" <type>
End of MapObjectList
```

Details

The mapObjectList command displays a list (by name) of the map objects of the specified type. See Examples for details.

There are four related commands that are used to get information about map objects: mapObjectTypeList, mapObjectTypeInfo, mapObjectList, and mapObjectInfo. These can be used in one of two ways:

- Exploratory - by getting broad/general information and "drilling down" to the desired specific information. For this method, you would:
  - Use mapObjectTypeList to show the map object <type>s.
  - Use mapObjectTypeInfo <type> to see if it has parameters or other information.
  - Use mapObjectList <type> to get the <name> of the map objects of that type.

  - Use mapObjectInfo <name> to get information about each map object (this is mostly for those that have parameters).
- Direct - by going after information on a specific map object. For this method, you would:
  - Use mapObjectInfo <name> to find out its <type> and its parameters.
  - Use mapObjectTypeInfo <type> to see what parameters it has and what they mean. This step isn't needed if you already know what the parameters mean. However, it can be useful for verifying ordering and other details.

For more details on these commands, see the links in the Related Commands section.

Examples

The following example lists the names of the "DriveOnRightSector" object types in the map:

```
mapobjectlist driveonrightsector
```

The command returns:

```
MapObjectList: "PreferredDirectionRightSingle1" DriveOnRightSector
MapObjectList: "PreferredDirectionRightSingle2" DriveOnRightSector
End of MapObjectList
```

Related Commands

mapObjectInfo Command on page 155

mapObjectTypeInfo Command on page 159

mapObjectTypeList Command on page 161

mapObjectTypeInfo Command

Gets detailed information about a particular type of map object.

Syntax

mapObjectTypeInfo <type>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

The command parameters are described in the following table.

Parameters	Definition
type	Enter a string that represents the type of objects. For example, SlowSector. The string must not contain spaces. The string must not be enclosed in double quotes.

Responses

The command returns:

```
MapObjectTypeList: <type> <metaType> "<label>" "<desc>"
MapObjectTypeInfoArgument: <argName> <argType> <argImportance> "<argDescription>"
MapObjectTypeInfoArgument: <argName> <argType> <argImportance> "<argDescription>"
MapObjectTypeInfoArgument: <argName> <argType> <argImportance> "<argDescription>"
End of MapObjectTypeInfo
```

Details

The mapObjectTypeInfo command displays detailed information about a specified type of map object. See Examples for details.

There are four related commands that are used to get information about map objects: mapObjectTypeList, mapObjectTypeInfo, mapObjectList, and mapObjectInfo. These can be used in one of two ways:

- Exploratory - by getting broad/general information and "drilling down" to the desired specific information. For this method, you would:
  - Use mapObjectTypeList to show the map object <type>s.
  - Use mapObjectTypeInfo <type> to see if it has parameters or other information.
  - Use mapObjectList <type> to get the <name> of the map objects of that type.

    - Use mapObjectInfo <name> to get information about each map object (this is mostly for those that have parameters).
- Direct - by going after information on a specific map object. For this method, you would:
    - Use mapObjectInfo <name> to find out its <type> and its parameters.
    - Use mapObjectTypeInfo <type> to see what parameters it has and what they mean. This step isn't needed if you already know what the parameters mean. However, it can be useful for verifying ordering and other details.

For more details on these commands, see the links in the Related Commands section.

Examples

The following example displays detailed information about the "DriveOnRightSector" object type:

```
mapObjectTypeInfo DriveOnRightSector
```

The command returns:

```
MapObjectTypeList: DriveOnRightSector SectorType "PreferredDirectionRightSingle" "One
Way Drive on Right"
MapObjectTypeInfoArgument: UseDefaultSideOffset bool Normal "True to use the default
side offset of 'Path Planning Settings'->'PreferredDirectionSideOffset', false to use
the PreferredDirectionSideOffset parameter of this object."
MapObjectTypeInfoArgument: PreferredDirectionSideOffset int Normal "The side offset for
this sector, which decides how far from the edge of the sector the robot will try to
drive. Setting this too low may cause the robot to pop out of the sector if it can get
to an open area."
End of MapObjectTypeInfo
```

Related Commands

mapObjectInfo Command on page 155

mapObjectList Command on page 157

mapObjectTypeList Command on page 161

mapObjectTypeList Command

Gets a list of the types of map objects in the map.

Syntax

mapObjectTypeList

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
MapObjectTypeList: <typeName> <metaType>
MapObjectTypeList: <typeName> <metaType>
End of MapObjectTypeList
```

Details

The mapObjectInfo command displays a list of the various types of map objects contained in the current map. See Examples for details.

There are four related commands that are used to get information about map objects: mapObjectTypeList, mapObjectTypeInfo, mapObjectList, and mapObjectInfo. These can be used in one of two ways:

- Exploratory - by getting broad/general information and "drilling down" to the desired specific information. For this method, you would:
  - Use mapObjectTypeList to show the map object <type>s.
  - Use mapObjectTypeInfo <type> to see if it has parameters or other information.
  - Use mapObjectList <type> to get the <name> of the map objects of that type.
  - Use mapObjectInfo <name> to get information about each map object (this is mostly for those that have parameters).
- Direct - by going after information on a specific map object. For this method, you would:
  - Use mapObjectInfo <name> to find out its <type> and its parameters.
  - Use mapObjectTypeInfo <type> to see what parameters it has and what they mean. This

step isn't needed if you already know what the parameters mean. However, it can be useful for verifying ordering and other details.

For more details on these commands, see the links in the Related Commands section.

Examples

The following example lists the types of map objects in the current map:

```
mapObjectTypeList
```

The command returns:

```
MapObjectTypeList: DriveOnRightSector SectorType
MapObjectTypeList: FastSector SectorType
MapObjectTypeList: LocalPathPlanningBehaviorSector SectorType
MapObjectTypeList: MovementParametersSector SectorType
End of MapObjectTypeList
```

Related Commands

mapObjectInfo Command on page 155

mapObjectList Command on page 157

mapObjectTypeInfo Command on page 159

newConfigParam Command

Adds a custom parameter to ARAM’s configuration, which can then be managed through ARCL or MobilePlanner.

Syntax

newConfigParam <section> <name> <description> <priority_level> <type> <default_value> <min> <max> <DisplayHint>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

The parameter is not persistent through an ARAM restart; however, its last-set value persists.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

The command parameters are described in the following table.

Parameters	Definition
section	Enter the name of the section where you want to add a new configuration parameter. This is a text string and is case-sensitive.
name	Enter the name of the new configuration parameter. This is a text string and is case-sensitive.
description	Enter a description of the new configuration parameter. This is a text string with quotes around it.
priority_level	Enter the priority level of the new parameter: Basic, Intermediate, Advanced, Expert or Factory.
type	Enter the type of parameter: integer, double, string, boolean or separator.
default_value	Enter the default value for the parameter.
min	Enter a minimum value, if applicable, otherwise enter "None".
max	Enter a maximum value, if applicable, otherwise enter "None".
DisplayHint	Enter a display hint for the new configuration parameter. This is a text string with quotes around it. If you do not want to use a display hint, enter "None".

Responses

The command returns:

```
Will add new param '<name>' to section '<section>'
```

Details

The newConfigParam command adds a custom parameter to ARAM’s configuration. After the parameter is added, it can be managed through ARCL or MobilePlanner. For details on managing parameters in MobilePlanner, see the *Adept Motivity User's Guide*.

Examples

The following example adds a new configuration parameter "newparam" to the section "Log":

```
newconfigparam Log newparam "this is a test param" Basic string "a test" none none "a
hint"
Will add new param 'newparam' to section 'Log'
```

You can see the new parameter by entering the getConfigSectionInfo command, as follows:

```
getconfigsectioninfo log
GetConfigSectionInfo: "" "CENTRAL_SECTION"
GetConfigSectionParamInfo: String newparam Basic None None "this is a test param" "a
hint"
EndOfGetConfigSectionInfo
```

Related Commands

configAdd Command on page 85

configParse Command on page 87

configStart Command on page 89

getConfigSectionInfo Command on page 117

getConfigSectionList Command on page 119

getConfigSectionValues Command on page 121

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

newConfigSectionComment Command

Adds a comment to a section.

Syntax

newConfigSectionComment <section> <comment>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

ARAM Settings

You have to explicitly enable this feature in MobilePlanner by checking and applying the ArclConfig parameter in the ARCL server setup section of the **Configuration > Robot Interface** tab. For more information, see Set ARCL Parameters in MobilePlanner on page 30. Changes do not take effect until: the robot is idle and stationary; the Configuration changes are saved.

Parameters

The command parameters are described in the following table.

Parameters	Definition
section	Enter the name of the section from which you want to see a list of parameter values. This is a text string and is case-sensitive.
comment	Enter a description of the new configuration parameter. This is a text string; quotes around it are optional.

Responses

The command returns:

```
Will add config comment '<comment>' to section '<section>'
```

Details

The newConfigSectionComment command allows you to enter a comment to display above the section's parameter list in the MobilePlanner configuration dialog.

Examples

This example adds the comment "my comments" to the section "Log":

```
newConfigSectionComment Log "my comments"
Will add config comment 'my comments' to section 'Log'
```

You can see the added comment by entering the getConfigSectionInfo command, as follows:

```
getconfigsectioninfo log
```

```
GetConfigSectionInfo: "my comments" "CENTRAL_SECTION"
GetConfigSectionParamInfo: String newparam Basic None None "this is a test param" "a
hint"
EndOfGetConfigSectionInfo
```

Related Commands

configAdd Command on page 85

configParse Command on page 87

configStart Command on page 89

getConfigSectionInfo Command on page 117

getConfigSectionList Command on page 119

getConfigSectionValues Command on page 121

newConfigParam Command on page 163

newConfigSectionComment Command on page 165

odometer Command

Shows the robot trip odometer readings.

Syntax

odometer

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Odometer: <distance> mm <heading> deg <time> sec
```

Details

How far and how long the robot has traveled since ARAM startup or reset. The odometer is reset with the odometerReset command. For details, see odometerReset Command on page 168.

Examples

To view the robot odometer readings, enter the following:

```
odometer
```

The command returns:

```
Odometer: 8281 mm 210 deg 469 sec
```

Related Commands

odometerReset Command on page 168

odometerReset Command

Resets the robot trip odometer.

Syntax

odometerReset

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Reset odometer
```

Details

The odometerReset command resets distance, heading and time odometer values to 0.

Examples

To reset the robot odometer, enter the following:

```
odometerreset
```

The command returns:

```
Reset odometer
```

Related Commands

odometer Command on page 167

oneLineStatus Command

Shows the status of the robot on one line of text.

Syntax

oneLineStatus

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Status: Arrived at <goal> BatteryVoltage: <volts_dc> Location: <X_mm> <Y_mm> <heading>
Temperature: <degrees>
```

Details

The oneLineStatus command returns the robot’s operating state, battery voltage and position status as a single line of text. To get a multi-line status of the robot, use the status command. For details, see status Command on page 276.

Examples

To get a one-line status of the robot, enter the following:

```
onelinestatus
```

The command returns:

```
Status: Arrived at g_24 BatteryVoltage: 13.0 Location: 7038 -8342 0 Temperature: -127
```

Related Commands

getDateTime Command on page 123

getGoals Command on page 124

getInfo Command on page 126

getInfoList Command on page 128

getPayload Command on page 132

getRoutes Command on page 134

queryDockStatus Command on page 203

queryMotors Command on page 207

status Command on page 276

outputList Command

Lists the named digital outputs.

Syntax

outputList

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Output: <name>
...
End of OutputList
```

Details

The outputList command returns the list of digital outputs. To get the status of a particular digital output, use the outputQuery command. For details, see outputQuery Command on page 175.

Examples

To get the list of digital outputs, enter the following:

```
outputlist
```

The command returns:

```
OutputList: out_one
OutputList: out_two
End of OutputList
```

Related Commands

inputList Command on page 142

inputQuery Command on page 144

outputOff Command on page 173

outputOn Command on page 174

outputQuery Command on page 175

outputOff Command

Turns off the named digital output.

Syntax

outputOff <name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter the name of the output to turn off.

Responses

The command returns:

```
Output: <name> <status>
```

Details

The outputOff command turns off the named digital output. To get a list of the digital outputs, use the outputList command. For details, see outputList Command on page 171.

Examples

To turn off digital output named "out_one", enter the following:

```
outputoff out_one
```

The command returns:

```
Output: out_one off
```

Related Commands

inputList Command on page 142

inputQuery Command on page 144

outputList Command on page 171

outputOn Command on page 174

outputQuery Command on page 175

outputOn Command

Turns on the named digital output.

Syntax

outputOn <name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter the name of the output to turn on.

Responses

The command returns:

```
Output: <name> <status>
```

Details

The outputOn command turns on the named digital output. To get a list of the digital outputs, use the outputList command. For details, see outputList Command on page 171.

Examples

To turn on digital output named "out_one", enter the following:

```
outputon out_one
```

The command returns:

```
Output: out_one on
```

Related Commands

inputList Command on page 142

inputQuery Command on page 144

outputList Command on page 171

outputOff Command on page 173

outputQuery Command on page 175

outputQuery Command

Queries the state of a named output.

Syntax

outputQuery <name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter the name of the output to query.

Responses

The command returns:

```
Output: <name> <status>
```

Details

The outputQuery command returns the status of the named digital output. To get a list of the digital outputs, use the outputList command. For details, see outputList Command on page 171.

Examples

To get the status of digital output named "out_one", enter the following:

```
outputquery out_one
```

The command returns:

```
Output: out_one off
```

Related Commands

inputList Command on page 142

inputQuery Command on page 144

outputList Command on page 171

outputOff Command on page 173

outputOn Command on page 174

patrol Command

Initiates continuous patrol of the named route.

Syntax

patrol <route_name>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
route_name	Enter the name of the route you want the robot to patrol.

Responses

The command returns:

```
Patrolling route <route_name>
```

Details

The patrol command instructs the robot to perform a continuous patrol of the named route. ("Patrol" means to stop at all the route goals in the order on the route list.) The robot will keep patrolling until a stop command is entered. For details, see stop Command on page 278.

Examples

The following example starts a patrol of the route named "test" and then interrupts the patrol with a stop command.

```
patrol test
Patrolling route test

stop
Interrupted: Patrolling route test
Stopping
Stopped
```

Related Commands

patrolOnce Command on page 179

patrolResume Command on page 181

stop Command on page 278

patrolOnce Command

Patrol the named route one time.

Syntax

patrolOnce <route_name> [index]

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
route_name	Enter the name of the route you want the robot to patrol.
index	Enter an optional index value. No value or 0 instructs the robot to start at the beginning of the route.

Responses

The command returns:

```
Patrolling route <route_name> once
Finished patrolling route <route_name>
```

Details

The patrolOnce command instructs the robot to patrol the named route one time. ("Patrol" means to stop at all the route goals in the order on the route list.) The patrol starts from the first goal on the list or from the specified indexed goal.

Examples

To command the robot to patrol the route "test", enter:

```
patrolonce test
```

The command returns:

```
Patrolling route test once
Finished patrolling route test
```

Related Commands

patrol Command on page 177

patrolResume Command on page 181

stop Command on page 278

patrolResume Command

Continue navigating the current route.

Syntax

patrolResume [route_name]

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
route_name	Enter the name of the route you want the robot to patrol.

Responses

The command returns:

```
Patrolling route <route_name> once
Finished patrolling route <route_name>
```

Details

The patrolResume command instructs the robot to continue the patrol of the named route. ("Patrol" means to stop at all the route goals in the order on the route list.)

Examples

The following example starts a patrol of the route named "test", interrupts the patrol with a stop command, and then uses the patrolResume command to continue the patrol.

```
patrolonce test 0
Patrolling route test once

stop
Interrupted: Patrolling route test once
Stopping
Stopped

patrolresume test
Patrolling route test once
Finished patrolling route test
```

Related Commands

patrol Command on page 177

patrolOnce Command on page 179

stop Command on page 278

pauseTaskCancel Command

Cancels the pause task if one is active.

Syntax

pauseTaskCancel

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
PauseTask: <status>
```

The pauseTaskCancel command returns one of the following status messages:

- PauseTask: Pausing with status "Pausing"
- PauseTask: Pausing interrupted
- PauseTask: Pausing cancelled
- PauseTask: Not pausing

These messages are broadcast to all of the clients, with the exception of "Not pausing".

Details

The pauseTaskCancel command is used to cancel a pause task if one is active. See the Examples section.

Examples

The following example starts, builds and executes a task list. The pauseTaskCancel command is used to end the "pause" (3rd) task on the list. When the pause task is canceled, the robot continues to the last goal (g_ 23).

```
liststart mylist
List being cleared
Making new list

listadd goto g_5
Added task 'goto g_5' to the list
listadd goto g_6
Added task 'goto g_6' to the list
listadd pause
Added task 'pause' to the list
listadd goto g_23
```

```
Added task 'goto g_23' to the list

listexecute
Executing list

pausetaskcancel
PauseTask: Pause cancelled
Successfully finished task list
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

pauseTaskState Command

Displays the status of the pause task.

Syntax

pauseTaskState

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
PauseState: <status>
```

The pauseTaskState command returns one of the following status messages:

- PauseState: Pausing with status "Pausing"
- PauseState: Pausing interrupted
- PauseState: Pausing cancelled
- PauseState: Not pausing

These messages are **not** broadcast to all of the clients, with the exception of "Not pausing". This command is helpful after a connection, to make sure a broadcast wasn't missed.

Examples

The following example shows the status of the pause task.

```
pausetaskstate
PauseState: Pausing with status "Pausing"
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

payloadQuery Command (shortcut: pq)

Queries the payload for a specified robot, a specified robot and slot, or all connected robots that have a payload configured.

Syntax

payloadQuery [robotName or "default"] [slotNumber or "default"] [echoString]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
robotName	Enter the name of the robot to display its slot information.
slotNumber	Enter the slot number to display its information. Requires a value in the previous parameter.
echoString	An optional string that is appended to each line of the results. Requires a value in the previous parameter.

Responses

The command returns the payload query in the following format:

```
PayloadQuery: "<robotName>" <slotNumber> "<description>" <date> <time> "[echoString]"
```

The date and time are assigned by the system when the slot payload is set on the robot.

Details

This command can be used to view the payload information for:

- all slots on all robots
- a specified slot on a robot
- all slots on a specified robot

Slot numbering starts at 1 (there is no slot 0).

An optional string can be specified, which will be appended to each line of the results.

Examples

In the example below, robot 21 is carrying books and glasses. To view what robot 21 is carrying, enter the following command:

```
payloadQuery 21
```

The command returns:

```
PayloadQuery: "21" 1 "Books" 05/07/2012 21:11:33 ""
PayloadQuery: "21" 2 "Glasses" 05/07/2012 21:15:11 ""
PayloadQuery: "21" 3 "Empty" None None ""
PayloadQuery: "21" 4 "Empty" None None ""
EndPayloadQuery
```

The following example displays all of the defined slots on all robots connected to the Enterprise Manager. The command is entered without the robotName argument.

```
payloadQuery
PayloadQuery: "21" 1 "Books" 05/07/2012 21:11:33 ""
PayloadQuery: "21" 2 "Glasses" 05/07/2012 21:14:51 ""
PayloadQuery: "21" 3 "Empty" None None ""
PayloadQuery: "21" 4 "Empty" None None ""
PayloadQuery: "22" 1 "Empty" None None ""
PayloadQuery: "22" 2 "Empty" None None ""
PayloadQuery: "22" 3 "stuff" 09/10/2012 12:14:14 ""
PayloadQuery: "22" 4 "Empty" None None ""
PayloadQuery: "23" 1 "morestuff" 09/10/2012 12:17:23 ""
PayloadQuery: "23" 2 "Empty" None None ""
PayloadQuery: "23" 3 "Bread" 09/10/2012 12:23:39 ""
PayloadQuery: "23" 4 "Empty" None None ""
EndPayloadQuery
```

The following example displays all of the defined slots on all robots and echoes the string "hello":

```
payloadquery default default hello
PayloadQuery: "31" 1 "slotjunk" 05/07/2012 21:11:33 hello
PayloadQuery: "31" 2 "abc" 05/07/2012 21:10:53 hello
PayloadQuery: "31" 3 "def" 09/10/2012 12:14:14 hello
PayloadQuery: "31" 4 "ghi" 09/10/2012 12:23:39 hello
PayloadQuery: "32" 1 "Empty" None None hello
PayloadQuery: "32" 2 "Empty" None None hello
PayloadQuery: "32" 3 "Empty" None None hello
PayloadQuery: "32" 4 "Empty" None None hello
PayloadQuery: "33" 1 "Empty" None None hello
PayloadQuery: "33" 2 "Empty" None None hello
PayloadQuery: "33" 3 "Empty" None None hello
PayloadQuery: "33" 4 "Empty" None None hello
PayloadQuery: "34" 1 "Empty" None None hello
PayloadQuery: "34" 2 "Empty" None None hello
PayloadQuery: "34" 3 "Empty" None None hello
PayloadQuery: "34" 4 "Empty" None None hello
PayloadQuery: "35" 1 "Empty" None None hello
PayloadQuery: "35" 2 "Empty" None None hello
PayloadQuery: "35" 3 "Empty" None None hello
```

```
PayloadQuery: "35" 4 "Empty" None None hello
PayloadQuery: "36" 1 "Empty" None None hello
PayloadQuery: "36" 2 "Empty" None None hello
PayloadQuery: "36" 3 "Empty" None None hello
PayloadQuery: "36" 4 "Empty" None None hello
EndPayloadQuery
```

Related Commands

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSet Command (shortcut: ps) on page 194

payloadSlotCount Command (shortcut: psc) on page 196

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPayload Command on page 272

payloadQueryLocal Command (shortcut: pql)

Queries the payload for the robot and specified slot.

Syntax

payloadQueryLocal [slotNumber or "default"] [echoString]

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
slotNumber	Enter the slot number to display its information.
echoString	An optional string that is appended to each line of the results. Requires a value in the previous parameter.

Responses

The command returns the payload query in the following format:

```
PayloadQueryLocal: <slotNumber> "<description>" <date> <time> "[echoString]"
```

The date and time are assigned by the system when the slot payload is set. For details, see payloadSet Command (shortcut: ps) on page 194.

Details

This command can be used to view the payload information for:

- all slots on the "default" robot
- a specified slot on the "default" robot

Slot numbering starts at 1 (there is no slot 0).

An optional string can be specified, which will be appended to each line of the results.

Examples

The following command displays all slots on the local robot and echoes the string "hello":

```
payloadquerylocal default hello
PayloadQuery: 1 "slotjunk" 05/07/2012 21:11:33 hello
PayloadQuery: 2 "abc" 05/07/2012 21:10:53 hello
```

```
PayloadQuery: 3 "def" 09/10/2012 12:14:14 hello
PayloadQuery: 4 "ghi" 09/10/2012 12:23:39 hello
EndPayloadQuery
```

Related Commands

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSet Command (shortcut: ps) on page 194

payloadSlotCount Command (shortcut: psc) on page 196

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPayload Command on page 272

payloadRemove Command (shortcut: pr)

Empties the specified payload slot on the robot.

Syntax

payloadRemove <slot_number>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
slot_number	Enter an integer greater than zero (slot numbering starts at 1).

Responses

The command returns the following for a pending item:

```
payloadremove attempting to remove slot <slot_number>
payloadremove on <robot> of slot number <slot_number> successfully
PayloadUpdate: "<robot>" <slot_number> "Empty" None None
```

Details

The payloadRemove command empties a payload slot on the robot. The slot number must be specified, and it starts at 1.

Examples

To empty payload slot 4 on the robot, enter

```
payloadRemove 4
```

The command returns:

```
payloadremove attempting to remove slot 4
payloadremove on 31 of slot number 4 successfully
PayloadUpdate: "31" 4 "Empty" None None
```

Related Commands

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadQuery Command (shortcut: pq) on page 187

payloadSet Command (shortcut: ps) on page 194

payloadSlotCount Command (shortcut: psc) on page 196

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPayload Command on page 272

payloadSet Command (shortcut: ps)

Defines a payload slot on this robot.

Syntax

payloadSet <slot_number> <slot_string>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
slot_number	Enter an integer greater than zero, to define a payload slot on this robot.
slot_string	Enter a description of the contents of the payload.

Responses

The command returns:

```
payloadset attempting to set payload <slot_number> "<slot_string>"
payloadset on "<robot>" of slot number <slot_number> with string "<slot_string>" suc-
cessfully set
PayloadUpdate: "<robot>" <slot_number> "<slot_string>"
```

Details

The payloadSet command defines a payload slot on the robot. These slots represent containers where the objects (payload) are carried on top of the robot. For example, you can assign a name to slot 1 on robot "xyz" that represents the object the robot is to carry from one goal to the next. This allows you to keep track of what the robot is transporting.

If the robot does not have multiple payload slots, you can use the setPayload command to set the payload name for the entire robot. For details, see setPayload Command on page 272.

Examples

To define payload slot 1 with the object "Books", enter:

```
payloadSet 1 Books
```

The command returns:

```
payloadset attempting to set payload 1 "Books"
payloadset on "Adept_Telepresence_Robot" of slot number 1 with string "Books" suc-
cessfully set
PayloadUpdate: "Adept_Telepresence_Robot" 1 "Books"
```

Related Commands

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSlotCount Command (shortcut: psc) on page 196

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPayload Command on page 272

payloadSlotCount Command (shortcut: psc)

Displays the slot count on a specific robot or on all robots.

Syntax

payloadSlotCount [robotName or "default"] [echoString]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
robotName	Enter the name of the robot to display its slot count. To view the slot counts for all connected robots, enter the command with no parameter or enter "default".
echoString	An optional string that is appended to each line of the results. Requires a value in the previous parameter.

Responses

The command returns the slot count in the following format:

```
PayloadSlotCount: "<robotName>" <slotCount> <date> <time> "[echoString]"
```

The date and time are assigned by the system.

Details

The payloadSlotCount command is used to display the slot count on a specific robot or on all robots. To limit the query to a specific robot, enter the robot name; to view the slot count on all robots, omit the robot name.

Slot numbering starts at 1 (there is no slot 0).

An optional string can be specified, which will be appended to each line of the results.

Examples

To view the slot count for robot 21, enter the following command:

```
payloadslotcount 21
```

The command returns:

```
PayloadSlotCount: "21" 4 ""
EndPayloadSlotCount
```

The following example displays the slot counts on all robots connected to the Enterprise Manager. The command is entered without the robotName argument.

```
payloadSlotCount
PayloadSlotCount: "21" 4 04/27/2012 06:37:33 “”
PayloadSlotCount: "22" 5 04/27/2012 08:37:33 “”
PayloadSlotCount: "23" 4 04/27/2012 07:37:33 “”
EndPayloadSlotCount
```

Related Commands

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSet Command (shortcut: ps) on page 194

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPayload Command on page 272

payloadSlotCountLocal Command (shortcut: pscl)

Displays a slot count on this robot.

Syntax

payloadslotcountlocal

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The payloadSlotCountLocal command does not have any arguments.

Examples

The following command displays the slot count for the local robot and echoes the string "testing":

```
payloadslotcountlocal
PayloadSlotCount: "Adept_Telepresence_Robot" 4
EndPayloadSlotCount
```

Related Commands

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSet Command (shortcut: ps) on page 194

payloadSlotCount Command (shortcut: psc) on page 196

setPayload Command on page 272

play Command

Plays a .wav sound file on the robot.

Syntax

play <path_file>

Usage Considerations

This ARCL command is only available on the robot.

The sound file must be in .wav format.

Parameters

The command parameters are described in the following table.

Parameter	Definition
path_file	Enter the path and name of the sound file with the .wav extension. Files in subfolders must be use a forward slash between folder names, for example: /subfolder1/subfolder2/wavefile.wav

Responses

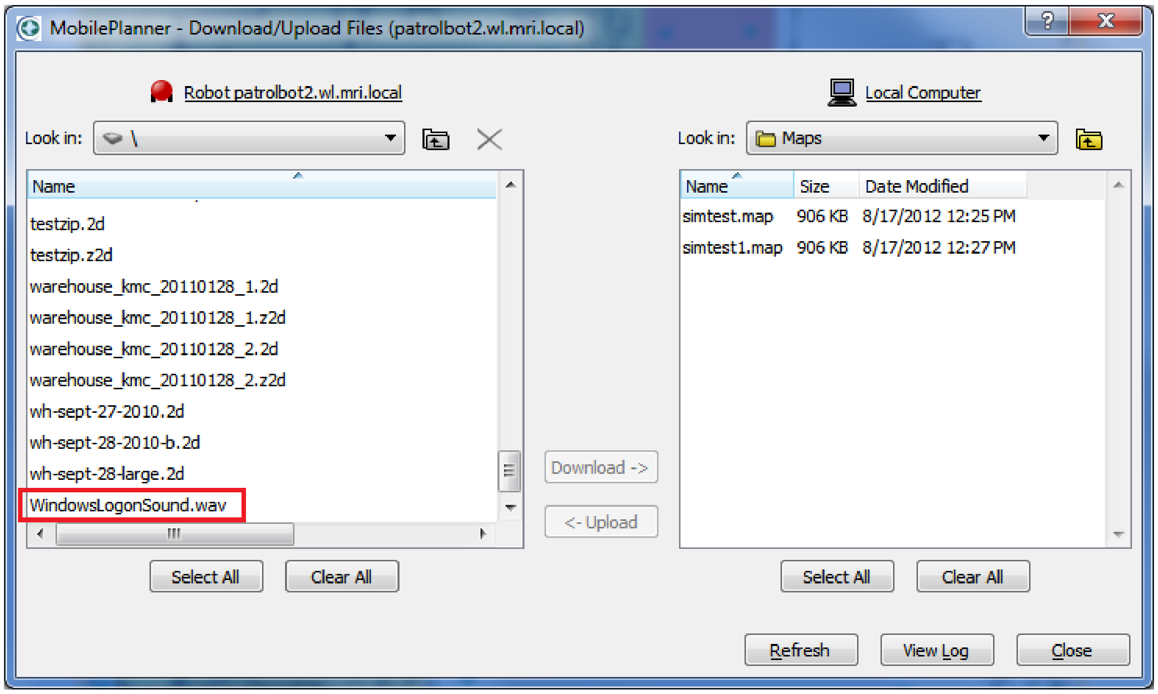
The command returns:

```
Playing <path_file>
```

Details

The play command plays a .wav sound file on the mobile robot. It is equivalent to the playInstant task, which plays the specified wave file through the robot's audio output, if enabled.

Although ARCL does not provide a way to list the sound files on the robot, you can view the files using MobilePlanner **File > Download/Upload** menu selection, as shown in the following figure.



Name
testzip.2d
testzip.z2d
warehouse_kmc_20110128_1.2d
warehouse_kmc_20110128_1.z2d
warehouse_kmc_20110128_2.2d
warehouse_kmc_20110128_2.z2d
wh-sept-27-2010.2d
wh-sept-28-2010-b.2d
wh-sept-28-large.2d
WindowsLogonSound.wav

Name	Size	Date Modified
simtest.map	906 KB	8/17/2012 12:25 PM
simtest1.map	906 KB	8/17/2012 12:27 PM

To have the robot speak a text string, use the say command. For details, see say Command on page 261.

Examples

The following example plays the file "WindowsLogonSound.wav", which is shown in the root folder of the robot in the previous figure.

```
play WindowsLogonSound.wav
Playing WindowsLogonSound.wav
```

Related Commands

say Command on page 261

popupSimple Command

Display a popup message in the MobileEyes software.

Syntax

popupSimple <"title"> <"message"> <"buttonLabel"> <timeout>

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

All parameters, except for timeout, must be enclosed in double quotes.

Parameters

The command parameters are described in the following table.

Parameter	Definition
"title"	Enter a string enclosed in double quotes for the title.
"message"	Enter a string enclosed in double quotes for the message.
"buttonLabel"	Enter a string enclosed in double quotes for the button label.
timeout	Integer that specifies the time (in seconds) the popup will remain on the screen.

Responses

The command returns:

```
Creating simple popup
```

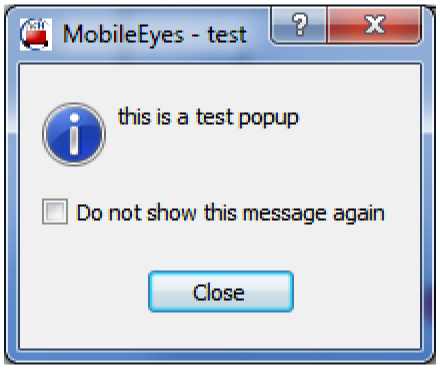
Details

The popupSimple command is used to create a popup message for the MobileEyes software. When the command is entered, the popup message is immediately displayed; it remains on screen for the timeout period (in seconds) or until the user clicks the button or close (x) icon, whichever occurs first.

Examples

The following example displays a simple popup test message, which remains on the screen for 30 seconds. A sample of the popup is shown in the following figure.

```
popupsimple "test" "this is a test popup" "Close" 30
Creating simple popup
```



Example Popup Message

Related Commands

play Command on page 199

say Command on page 261

queryDockStatus Command

Gets the docking/charging status.

Syntax

queryDockStatus

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
DockingState: <Docking,Docked,Undocked> ForcedState: <Forced,Unforced> ChargeState:
<Not,Bulk,Overcharge,Float>
```

Details

The queryDockStatus command returns the current docking/charging state of the robot.

Examples

To view the robot docking/charge status, enter:

```
querydockstatus
```

The command returns:

```
DockingState: Docking ForcedState: Unforced ChargeState: Not
```

Related Commands

queryMotors Command on page 207

status Command on page 276

queryFaults Command (shortcut: qf)

Displays the faults associated with the specified robot.

Syntax

queryFaults [robotName or "default"] [echoString]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Displays all faults on the specified robot. Displays faults on all robots if the robotName parameter is omitted.

Parameter

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
robotName	Enter the name of the robot. To view all the robots connected to the Enterprise Manager, omit this parameter or enter "default".
echoString	An optional string that is appended to each line of the results.

Responses

The command returns the following for a pending item:

```
RobotFaultQuery: <robotName> <faultName> <faultShortDescription> <faultLongDescription>
<bool:drivingFault> <bool:criticalFault><bool:applicationFault><bool:clearedOnGo><bool:
clearedOnAcknowledgement> <echoString>
EndQueryFaults
```

Details

The queryFaults command provides a listing of all faults for the specified robot, or all faults for all robots connected to the Enterprise Manager if no robot is specified.

Example

```
queryfaults robot1
RobotFaultQuery: "robot1" Fault_Critical_Application fault1 "shortdesc" "longdesc" false
true true false false “”
EndQueryFaults

queryfaults robot1 echoit
RobotFaultQuery: "robot1" Fault_Critical_Application fault1 "shortdesc" "longdesc" false
```

```
true true false false echoit
EndQueryFaults

queryfaults
RobotFaultQuery: "robot2" Fault_Driving_Application fault2 "shortd" "longd" true false
true false false “”
RobotFaultQuery: "robot1" Fault_Critical_Application fault1 "shortdesc" "longdesc" false
true true false false “”
EndQueryFaults

queryfaults
RobotFaultQuery: "guiabot_2010_09_20" Fault_Driving_Application fault2 "shortd" "longd"
true false true false false “”
RobotFaultQuery: "showpatrolbot1" Fault EncoderDegraded "Encoder degraded" "The robot's
encoders may be degraded" false true false false false “”
RobotFaultQuery: "showpatrolbot1" Fault_Driving EncoderFailed "Encoder failed" "The
robot's encoders have failed, turn off the robot and contact your robot provider for main-
tenance" true true false false false “”
RobotFaultQuery: "showpatrolbot1" Fault_Critical GyroFault "Gyro fault" "The robot's gyro
has had a critical fault, you may power cycle the robot and continue using it, but you
should also contact your robot provider for maintenance" true true false false false “”
RobotFaultQuery: "showpatrolbot1" Fault_Critical OverTemperatureAnalog "Robot overheated
(analog)" "The robot is too hot (measured by analog) and will shut down shortly" false
true false false false “”
RobotFaultQuery: "showpatrolbot1" Fault_Critical UnderVoltage "Robot battery critically
low" "The robot battery is critically low and will shut down shortly" false true false
false false “”
RobotFaultQuery: "showpatrolbot1" Fault_Critical_Application fault1 "shortdesc" "long-
desc" false true true false false “”
RobotFaultQuery: "showpatrolbot1" Fault_Application fault3 "short" "long" false true true
false false “”
EndQueryFaults
```

The broadcast messages to EM ARCL when robots set/clear faults will have the following formats:

```
RobotFault: " showpatrolbot1" Fault_Application fault3 "short" "long" false true true
false false
RobotFault: " showpatrolbot1" Fault_Driving_Application fault2 "shortd" "longd" true
false true false false
RobotFault: " showpatrolbot1" Fault_Critical OverTemperatureAnalog "Robot overheated (ana-
log)" "The robot is too hot (measured by analog) and will shut down shortly" false true
false false false
RobotFault: " showpatrolbot1" Fault_Critical UnderVoltage "Robot battery critically low"
"The robot battery is critically low and will shut down shortly" false true false false
false
RobotFault: " showpatrolbot1" Fault EncoderDegraded "Encoder degraded" "The robot's
encoders may be degraded" false true false false false
RobotFault: " showpatrolbot1" Fault_Driving EncoderFailed "Encoder failed" "The robot's
encoders have failed, turn off the robot and contact your robot provider for maintenance"
true true false false false
RobotFault: " showpatrolbot1" Fault_Critical GyroFault "Gyro fault" "The robot's gyro has
had a critical fault, you may power cycle the robot and continue using it, but you should
also contact your robot provider for maintenance" true true false false false
RobotFault: "Sim2" Fault_Application_ClearedOnAcknowledgement f1 "s" "l" false false true
```

```
false true
RobotFaultCleared: "showpatrolbot1" Fault EncoderDegraded "Encoder degraded" "The
robot's encoders may be degraded" false true false false false
RobotFaultCleared: "showpatrolbot1" Fault_Driving EncoderFailed "Encoder failed" "The
robot's encoders have failed, turn off the robot and contact your robot provider for
maintenance" true true false false false
RobotFaultCleared: "showpatrolbot1" Fault_Critical GyroFault "Gyro fault" "The robot's
gyro has had a critical fault, you may power cycle the robot and continue using it, but
you should also contact your robot provider for maintenance" true true false false false
RobotFaultCleared: "showpatrolbot1" Fault_Critical OverTemperatureAnalog "Robot over-
heated (analog)" "The robot is too hot (measured by analog) and will shut down shortly"
false true false false false
RobotFaultCleared: "showpatrolbot1" Fault_Critical UnderVoltage "Robot battery crit-
ically low" "The robot battery is critically low and will shut down shortly" false true
false false false
RobotFaultCleared: "showpatrolbot1" Fault_Critical_Application fault1 "shortdesc" "long-
desc" false true true false false
RobotFaultCleared: "Sim2" Fault_Application_ClearedOnAcknowledgement f1 "s" "l" false
false true false true
EndQueryFaults
```

Related Commands

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueModify Command (shortcut: qmod) on page 218

queueModify Command (shortcut: qmod) on page 218

queueMulti Command (shortcut: qm) on page 229

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShowRobot Command (shortcut: qsr) on page 251

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queryMotors Command

Gets the state of the robot motors.

Syntax

queryMotors

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Motors <enabled or disabled>
```

or

```
Estop pressed
```

or

```
Estop relieved but motors still disabled
```

Details

The queryMotors command returns the current state of the robot motors. The response includes motors enable and e-stop status.

Examples

To view the current state of the robot motors, enter:

```
querymotors
```

The command returns:

```
Motors enabled
```

With an Estop event:

```
EStop pressed
EStop relieved but motors still disabled
Motors enabled
Stopping
Stopped
```

Here's one with queries colorcoded (red == broadcasts, black are my queries and the responses to them):

```
EStop pressed

querymotors
EStop pressed

EStop relieved but motors still disabled

querymotors
EStop relieved but motors still disabled
Motors enabled
Stopping
Stopped

querymotors
Motors enabled
```

The motor disabled won't normally be seen (the old robots had buttons to do that, the new ones don't)... but you can see it with the 4 0 above. It's:

```
Motors disabled
Motors enabled
Stopping
Stopped
```

With queries colorcoded (red == broadcasts, black are my queries and the responses to them):

```
Motors disabled

querymotors
Motors disabled

Motors enabled

Stopping
Stopped

querymotors
Motors enabled
```

The 'Stopping' and 'Stopped' shouldn't be mentioned because other things could happen there (if a robot was docked, or if it had a pending job or something, it'd be different messages).

Related Commands

queryDockStatus Command on page 203

status Command on page 276

queueCancel Command (shortcut: qc)

Cancels a queued request for a robot by type or value.

Syntax

queueCancel <type> <value> [echoString or "default"] [reason]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
type	Enter the type of job. Valid types are: • id = the pickup or dropoff identification • jobId = the job identification • robotName = the robot name • status = the item status.
value	Enter the value that corresponds with the type used: For id, enter the pickup or dropoff identification, for example: PICKUP2 For jobId, enter the job identification, for example: JOB2 For robotName, enter the robot name, for example: robot_34 For status, enter one of the following values: • inprogress = cancels a job with an InProgress status. • pending = cancels a job with a Pending status. • interrupted = cancels a job with an Interrupted status.
echoString	An optional string that is appended to each line of the results.
reason	An optional string that can be used to provide a reason for the cancellation.

Responses

The command returns the following for a pending item:

```
queuecancel cancelling <cancelType> <cancelValue> <echoString> <reason> from queue
```

```
QueueUpdate: <id> <jobId> <priority> <status = Cancelled> <subStatus = reason_or_None>
Goal <"goalName"> <”robotName”>  <queuedDate> <queuedTime> <completedDate> <com-
pletedTime> <echoString>
```

The command returns the following for an in-progress item:

```
queuecancel cancelling <cancelType> <cancelValue> <echoString> from queue
QueueUpdate: <id> <jobId> <priority> <status = Cancelling> <subStatus = reason_or_None>
Goal <"goalName"> <”robotName”>  <queuedDate> <queuedTime> <completedDate = None> <com-
pletedTime = None> <echoString>
QueueUpdate: <id> <jobId> <priority> <status = Interrupted> <subStatus = reason_or_None>
Goal <"goalName"> <”robotName”>  <queuedDate> <queuedTime> <completedDate = None> <com-
pletedTime = None> <failedCount>
QueueUpdate: <id> <jobId> <priority> <status = Cancelled> <subStatus = reason_or_None>
Goal <"goalName"> <”robotName”>  <queuedDate> <queuedTime> <completedDate> <com-
pletedTime> <failedCount>
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

Details

The queueCancel command is used to cancel a queued robot request. The request can be canceled by type (such as the robot name or job identification) or by the request status.

An optional string can be specified, which will be appended to each line of the results.

Examples

In the following example, a pending item in the queue is canceled.

```
queuepickup x
queuepickup goal "x" with priority 10, id PICKUP1 and jobId JOB1 successfully queued
QueueUpdate: PICKUP1 JOB1 10 Pending None Goal "x" "None" 04/15/2015 6:32:47 None None 0
queuecancel jobid job1
QueueUpdate cancelling "jobid" "job1" "" "None" from queue
QueueUpdate: PICKUP1 JOB1 10 Cancelled None Goal "x" "None" 04/15/2015 6:32:47
04/15/2015 6:32:53 ""
```

In the following example, a request that is in progress is canceled.

```
QueueUpdate: PICKUP8 JOB8 10 InProgress None Goal "w20" MT-490 12/16/2014 13:19:07 None
None
queuecancel goal w20 abc
QueueUpdate: PICKUP8 JOB8 10 Cancelling None Goal "w20" None 12/16/2014 13:19:07 None
None abc
QueueUpdate: PICKUP8 JOB8 10 Interrupted None Goal "w20" None 12/16/2014 13:19:07 None
None
QueueUpdate: PICKUP8 JOB8 10 Cancelled None Goal "w20" None 12/16/2014 13:19:07
12/16/2014 13:19:13
```

In the following example, a request that is in progress is canceled. The cancel request includes a reason for the cancellation but no echo.

```
QueueUpdate: PICKUP2 JOB2 10 InProgress After Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:04:59 None None 0

queuecancel id pickup2 default reason

queuecancel cancelling "id" "pickup2" "" "reason" from queue
QueueUpdate: PICKUP2 JOB2 10 Cancelling reason Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:04:59 None None ""
QueueUpdate: PICKUP2 JOB2 10 Interrupted None Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:04:59 None None 0
QueueUpdate: PICKUP2 JOB2 10 Cancelled reason Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:04:59 01/21/2014 15:05:40 0
```

In the following example, a request that is in progress is canceled. The cancel request includes no reason for the cancellation and no echo.

```
QueueUpdate: PICKUP3 JOB3 10 InProgress After Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:07:58 None None 0

queuecancel jobid job3

QueueUpdate cancelling "jobid" "job3" "" "None" from queue
QueueCancel: PICKUP3 JOB3 10 Cancelling None Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:07:58 None None ""
QueueUpdate: PICKUP3 JOB3 10 Interrupted None Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:07:58 None None 0
QueueUpdate: PICKUP3 JOB3 10 Cancelled None Goal "w20" "guiabot_2010_09_20" 01/21/2014
15:07:58 01/21/2014 15:08:32 0
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueCancelLocal Command (shortcut: qcl)

Cancels a queued request for a robot by type or value.

Syntax

queueCancelLocal <type> <value> [echoString] [reason]

Usage Considerations

This ARCL command is only available on the robot.

Because the queueCancelLocal command is only available on the robot, it assumes it applies only to the items queued for that robot. This is a powerful difference (and feature) of the "local" version of the command. So, for example, a "queueCancelLocal status inprogress" command would allow you to cancel, based on inprogress status, all jobs queued for that particular robot.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
type	Enter the type of job. Valid types are: • id = the pickup or dropoff identification • jobId = the job identification • robotName = the robot name • status = the item status.
value	Enter the value that corresponds with the type used: For id, enter the pickup or dropoff identification, for example: PICKUP2 For jobId, enter the job identification, for example: JOB2 For status, enter one of the following values: • inprogress = queries a job with an InProgress status. • pending = queries a job with a Pending status. • interrupted = queries a job with an Interrupted status. **NOTE**: The value is ignored if type is <robotname>.
echoString	An optional string that is appended to each line of the results.
reason	An optional string that can be used to provide a reason for the cancellation.

Responses

The command returns the following for a pending item:

```
queuecancel cancelling <cancelType> <cancelValue> <echoString> <reason> from queue
QueueUpdate: <id> <jobId> <priority> <status = Cancelled> <subStatus = reason_or_None>
Goal <"goalName"> <"robotName">  <queuedDate> <queuedTime> <completedDate> <com-
pletedTime> <echoString>
```

The command returns the following for an in-progress item:

```
queuecancel cancelling <cancelType> <cancelValue> <echoString> from queue
QueueUpdate: <id> <jobId> <priority> <status = Cancelling> <subStatus = reason_or_None>
Goal <"goalName"> <"robotName">  <queuedDate> <queuedTime> <completedDate = None> <com-
pletedTime = None> <echoString>
QueueUpdate: <id> <jobId> <priority> <status = Interrupted> <subStatus = reason_or_None>
Goal <"goalName"> <"robotName">  <queuedDate> <queuedTime> <completedDate = None> <com-
pletedTime = None> <failedCount>
QueueUpdate: <id> <jobId> <priority> <status = Cancelled> <subStatus = reason_or_None>
Goal <"goalName"> <"robotName">  <queuedDate> <queuedTime> <completedDate> <com-
pletedTime> <failedCount>
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

Details

Because the queueCancelLocal command is only available on the robot, it assumes it applies only to the items queued for that robot. This is a powerful difference (and feature) of the "local" version of the command. So, for example, a "queueCancelLocal status inprogress" command would allow you to cancel, based on inprogress status, all jobs queued for that particular robot.

An optional string can be specified, which will be appended to each line of the results.

Example

The following example uses cancellocal with robotname (Note: robotname value field is ignored).

```
queuecancellocal robotname
queuecancel attempting to cancel "robotname" "Bullwinkle-[.53]" "" "None"
queuecancel cancelling "robotname" "Bullwinkle-[.53]" "" "None" from queue

QueueCancel: DROPOFF18 JOB18 20 Cancelling None Goal "w20" "Bullwinkle-[.53]" 01/21/2014
15:15:30 None None ""
QueueUpdate: DROPOFF18 JOB18 20 Interrupted None Goal "w20" "Bullwinkle-[.53]" 01/21/2014
15:15:30 None None 0
QueueUpdate: DROPOFF18 JOB18 20 Cancelled None Goal "w20" "Bullwinkle-[.53]" 01/21/2014
15:15:30 01/21/2014 15:16:07 0
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueDropoff Command (shortcut: qd)

Queues the robot to the dropoff goal.

Syntax

queueDropoff <goalName> [priority] [jobId]

Usage Considerations

This ARCL command is only available on the robot.

ARAM Settings

In order to use this feature, you have to explicitly enable it in the MobilePlanner software, by setting the EnterpriseQueuing argument in the Enterprise Features section of the **Configuration > Enterprise** tab.

Parameters

The queueDropoff arguments are described in the table below.

For details on the data types, see Data Types on page 48.

Parameter	Definition
goalName	Enter the name of the goal where you want the mobile robot to make a delivery.
priority	Enter an optional integer value that represents the priority of the dropoff request. The higher the number, the sooner Enterprise Manager is going to service the item. The default priority is 10, which can be changed in MobilePlanner.
jobId	Enter an optional identifier for the specified job. You can use a combination of string characters and integers. The jobId is helpful in tracking the job. If nothing is entered, ARCL generates a random jobId.

Responses

The command returns:

```
queuedropoff attempting to queue goal <goalName> <priority> <jobId>
queuedropoff goal <goalName> with priority <priority> id <id> and job_id <jobId> suc-
cessfully queued
QueueUpdate: <id> <jobId> <priority> <status> <substatus> Goal <goalName>  <robotName>
<queuedDate> <queuedTime> <completedDate> <completedTime> <failedCount>
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

Details

The queueDropoff command tells the mobile robot to go to a specified goal, typically to make a delivery.

Examples

The following example shows a queuedropoff at goal x with priority 22, job_id y4rt.

```
queuedropoff x 22 y4rt
queuedropoff attempting to queue goal "x" with priority 22
queuedropoff goal "x" with priority 22, id DROPOFF18 and job_id y4rt successfully queued
QueueUpdate: DROPOFF18 y4rt 22 Pending None Goal "x" “MT-490” 12/19/2011  07:07:53 None
None 0
Going to X
QueueUpdate: DROPOFF18 y4rt 22 InProgress UnAllocated Goal "x" “MT-490” 12/19/2011
07:07:53 None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress Allocated Goal "x" “MT-490” 12/19/2011
07:07:53 None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress BeforeDropoff Goal "x" “MT-490” 12/19/2011
07:07:53 None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress BeforeEvery Goal "x" “MT-490” 12/19/2011
07:07:53 None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress Before Goal "x" “MT-490” 12/19/2011  07:07:53
None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress Driving Goal "x" “MT-490” 12/19/2011  07:07:53
None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress After Goal "x" “MT-490” 12/19/2011  07:07:53
None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress AfterEvery Goal "x" “MT-490” 12/19/2011
07:07:53 None None 0
QueueUpdate: DROPOFF18 y4rt 22 InProgress AfterPickup Goal "x" “MT-490” 12/19/2011
07:07:53 None None 0
Arrived at X
QueueUpdate: DROPOFF18 y4rt 22 Completed None Goal "x" “MT-490” 01/19/2011  07:07:53
01/19/2011  07:08:07 0
```

Related Commands

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241 queueShow Command (shortcut: qs) on page 247

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueModify Command (shortcut: qmod)

Allows modification of goal and priority for job segments in these job types:

- PickupDropoff
- Pickups
- Dropoffs
- Swaps
- QueueMulti

Allows modification of segments in these states:

- Pending job segments
- InProgress jobs up to and including "InProgressDriving", but not after

Changing the priority for the first segment in a job may change the order in which it gets assigned. Changing the priority of other segments in the job will never change the order in which the job is assigned.

The queue time for a job will never be changed as a result of a queueModify command,

Changing the shared goal in a swap will break the link between the two jobs. Changing the other goals in the swap will not break the link.

Modified jobs will be candidates for swaps. The linking would occur immediately following the modify

Syntax

queueModify <id> <type> <value>

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

ARAM Settings

In order to use this feature, you have to explicitly enable it in the MobilePlanner software, by setting the EnterpriseQueuing argument in the Enterprise Features section of the **Configuration > Enterprise** tab.

Parameters

The queueModify arguments are described in the table below.

For details on the data types, see Data Types on page 48.

Parameter	Definition
<id>	Enter the string id for the job segment you wish to modify (either PICKUPxx or DROPOFFxx)
<type>	Enter the type of modification. Valid types are: • goal = the goal identification • priority = the priority level
<value>	Enter the value that corresponds with the type used: For goal, enter the goal identification, for example: goal_1 For priority, enter the priority level, for example: 10

Responses

Returns (for goal modify of a pending item)

```
queuemodify modifying id <id> goal <”modifiedGoal”>
QueueUpdate: <id> <jobId> <priority> BeforeModify None Goal <goal> "None" <queuedDate>
<queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> AfterModify None Goal <modifiedGoal> "None"
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> Pending None Goal <modifiedGoal> "None" <queuedDate>
<queuedTime> None None 0
```

Returns (for priority modify of a pending item)

```
queuemodify modifying id <id> priority <modifiedpriority>
QueueUpdate: <id> <jobId> <priority> BeforeModify None Goal <goal> "None" <queuedDate>
<queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> AfterModify None Goal <goal> "None"
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> Pending None Goal <modifiedGoal> "None"
<queuedDate> <queuedTime> None None 0
```

Returns (for goal modify of an in-progress item)

```
queuemodify modifying id <id> goal <modifiedGoal>
QueueUpdate: <id> <jobId> <priority> BeforeModify Driving Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> InterruptedByModify None Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> AfterModify None Goal <modifiedGoal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> Pending None Goal <modifiedGoal> "None" <queuedDate>
<queuedTime> None None 0
```

Returns (for priority modify of an in-progress item)

```
queuemodify modifying id <id> priority <modifiedPriority>
QueueUpdate: <id> <jobId> <priority> BeforeModify Driving Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> InterruptedByModify None Goal <goal> <robot>
```

```
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> AfterModify None Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> Pending None Goal <goal> "None"
<queuedDate> <queuedTime> None None 0
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

Details

The queueModify command allows modification of goal or priority values for job segments in these job types:

- Pickup-dropoff
- Pickups
- QueueMulti

It allows modification of segments in these states:

- Pending job segments
- InProgress jobs up to and including "InProgress Driving", but not after

Changing the priority for the first segment in a job may change the order in which it gets assigned. Changing the priority of other segments in the job will never change the order in which the job is assigned.

The queue time for a job will never be changed as a result of a queueModify command.

Changing the shared goal in a swap will break the link between the two jobs. Changing the other goals in the swap will not break the link.

Modified jobs will be candidates for swaps. The linking would occur immediately following the modify.

Examples

Example #1 – goal modify of a pending item:

```
queuePickup t
queuePickup goal "t" with priority 10 id PICKUP5 and jobId JOB5 successfully queued
QueueUpdate: PICKUP5 JOB5 10 Pending None Goal "t" "None" 03/25/2015 07:36:58 None None
0
queuemodify pickup5 goal w20
queuemodify modifying id pickup5 goal "w20"
QueueUpdate: PICKUP5 JOB5 10 BeforeModify None Goal "t" "None" 03/25/2015 07:36:58 None
None 0
QueueUpdate: PICKUP5 JOB5 10 AfterModify None Goal "w20" "None" 03/25/2015 07:36:58 None
None 0
QueueUpdate: PICKUP5 JOB5 10 Pending None Goal "w20" "None" 03/25/2015 07:36:58 None
None 0

queueDropoff y
queueDropoff attempting to queue goal "y" using default priority
queueDropoff goal "y" with priority 20 id DROPOFF6 and jobId JOB6 successfully queued
QueueUpdate: DROPOFF6 JOB6 20 Pending None Goal "y" "robotOne" 03/25/2015 07:38:09 None
```

```
None 0
queuemodifylocal dropoff6 goal x
queuemodifylocal modifying id dropoff6 goal "x"
QueueUpdate: DROPOFF6 JOB6 20 BeforeModify None Goal "y" "robotOne" 03/25/2015 07:38:09
None None 0
QueueUpdate: DROPOFF6 JOB6 20 AfterModify None Goal "x" "robotOne" 03/25/2015 07:38:09
None None 0
QueueUpdate: DROPOFF6 JOB6 20 Pending None Goal "x" "robotOne" 03/25/2015 07:38:09 None
None 0
```

Example #2 – priority modify of a pending item:

```
queueDropoff w20
queueDropoff attempting to queue goal "w20" using default priority
queueDropoff goal "w20" with priority 20 id DROPOFF7 and jobId JOB7 successfully queued
QueueUpdate: DROPOFF7 JOB7 20 Pending None Goal "w20" "robotOne" 03/25/2015 07:39:01 None
None 0
queuemodifylocal dropoff7 priority 22
queuemodifylocal modifying id dropoff7 priority 22
QueueUpdate: DROPOFF7 JOB7 20 BeforeModify None Goal "w20" "robotOne" 03/25/2015 07:39:01
None None 0
QueueUpdate: DROPOFF7 JOB7 22 AfterModify None Goal "w20" "robotOne" 03/25/2015 07:39:01
None None 0
QueueUpdate: DROPOFF7 JOB7 22 Pending None Goal "w20" "robotOne" 03/25/2015 07:39:01 None
None 0

queuePickup v
queuePickup goal "v" with priority 10 id PICKUP8 and jobId JOB8 successfully queued
QueueUpdate: PICKUP8 JOB8 10 Pending None Goal "v" "None" 03/25/2015 07:40:24 None None 0
queuemodify pickup8 priority 6
queuemodify modifying id pickup8 priority 6
QueueUpdate: PICKUP8 JOB8 10 BeforeModify None Goal "v" "None" 03/25/2015 07:40:24 None
None 0
QueueUpdate: PICKUP8 JOB8 6 AfterModify None Goal "v" "None" 03/25/2015 07:40:24 None
None 0
QueueUpdate: PICKUP8 JOB8 6 Pending None Goal "v" "None" 03/25/2015 07:40:24 None None 0
```

Example #3 – goal modify of an inProgress item:

```
queuePickup x
queuePickup goal "x" with priority 10 id PICKUP9 and jobId JOB9 successfully queued
QueueUpdate: PICKUP9 JOB9 10 Pending None Goal "x" "None" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress UnAllocated Goal "x" "robotTwo" 03/25/2015
07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Allocated Goal "x" "robotTwo" 03/25/2015 07:47:21
None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Driving Goal "x" "robotTwo" 03/25/2015 07:47:21
None None 0
queuemodify pickup9 goal y
queuemodify modifying id pickup9 goal "y"
QueueUpdate: PICKUP9 JOB9 10 BeforeModify Driving Goal "x" "robotTwo" 03/25/2015 07:47:21
None None 0
QueueUpdate: PICKUP9 JOB9 10 InterruptedByModify None Goal "x" "robotTwo" 03/25/2015
07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 AfterModify None Goal "y" "robotTwo" 03/25/2015 07:47:21
```

```
None None 0
QueueUpdate: PICKUP9 JOB9 10 Pending None Goal "y" "None" 03/25/2015 07:47:21 None None
0
QueueUpdate: PICKUP9 JOB9 10 InProgress UnAllocated Goal "y" "robotTwo" 03/25/2015
07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Allocated Goal "y" "robotTwo" 03/25/2015
07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Driving Goal "y" "robotTwo" 03/25/2015 07:47:21
None None 0
QueueUpdate: PICKUP9 JOB9 10 Completed None Goal "y" "robotTwo" 03/25/2015 07:47:21
03/25/2015 07:48:00 0
```

Example #4 – priority modify of an inProgress item:

```
queuePickup t
queuePickup goal "t" with priority 10 id PICKUP10 and jobId JOB10 successfully queued
QueueUpdate: PICKUP10 JOB10 10 Pending None Goal "t" "None" 03/25/2015 07:49:34 None
None 0
QueueUpdate: PICKUP10 JOB10 10 InProgress UnAllocated Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 10 InProgress Allocated Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 10 InProgress Driving Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
queuemodify pickup10 priority 13
queuemodify modifying id pickup10 priority 13
QueueUpdate: PICKUP10 JOB10 10 BeforeModify Driving Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 10 InterruptedByModify None Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 AfterModify None Goal "t" "robotTwo" 03/25/2015 07:49:34
None None 0
QueueUpdate: PICKUP10 JOB10 13 Pending None Goal "t" "None" 03/25/2015 07:49:34 None
None 0
QueueUpdate: PICKUP10 JOB10 13 InProgress UnAllocated Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 InProgress Allocated Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 InProgress Driving Goal "t" "robotTwo" 03/25/2015
07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 Completed None Goal "t" "robotTwo" 03/25/2015 07:49:34
03/25/2015 07:49:46 0
```

Related Commands

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueMulti Command (shortcut: qm) on page 229

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queryFaults Command (shortcut: qf) on page 204

queueModifyLocal Command (shortcut: qmodl)

Allows modification of goal and priority for job segments in these job types:

- Dropoffs
- Swaps

Allows modification of segments in these states:

- Pending job segments
- InProgress jobs up to and including "InProgressDriving", but not after

Syntax

queueModifyLocal <id> <type> <value>

Usage Considerations

This ARCL command is only available on the robot.

Because the queueModifyLocal command is only available on the robot, it assumes it applies only to the items queued for that robot. This is a powerful difference (and feature) of the "local" version of the command.

ARAM Settings

In order to use this feature, you have to explicitly enable it in the MobilePlanner software, by setting the EnterpriseQueuing argument in the Enterprise Features section of the **Configuration > Enterprise** tab.

Parameters

The queueModifyLocal arguments are described in the table below.

For details on the data types, see Data Types on page 48.

Parameter	Definition
<id>	Enter the string id for the job segment you wish to modify (either PICKUPxx or DROPOFFxx)
<type>	Enter the type of modification. Valid types are: • goal = the goal identification • priority = the priority level
<value>	Enter the value that corresponds with the type used: For goal, enter the goal identification, for example: goal_1 For priority, enter the priority level, for example: 10

Responses

Returns (for goal modify of a pending item)

```
queuemodify modifying id <id> goal <”modifiedGoal”>
QueueUpdate: <id> <jobId> <priority> BeforeModify None Goal <goal> "None" <queuedDate>
<queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> AfterModify None Goal <modifiedGoal> "None"
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> Pending None Goal <modifiedGoal> "None" <queuedDate>
<queuedTime> None None 0
```

Returns (for priority modify of a pending item)

```
queuemodify modifying id <id> priority <modifiedpriority>
QueueUpdate: <id> <jobId> <priority> BeforeModify None Goal <goal> "None" <queuedDate>
<queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> AfterModify None Goal <goal> "None"
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> Pending None Goal <modifiedGoal> "None"
<queuedDate> <queuedTime> None None 0
```

Returns (for goal modify of an in-progress item)

```
queuemodify modifying id <id> goal <modifiedGoal>
QueueUpdate: <id> <jobId> <priority> BeforeModify Driving Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> InterruptedByModify None Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> AfterModify None Goal <modifiedGoal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> Pending None Goal <modifiedGoal> "None" <queuedDate>
<queuedTime> None None 0
```

Returns (for priority modify of an in-progress item)

```
queuemodify modifying id <id> priority <modifiedPriority>
QueueUpdate: <id> <jobId> <priority> BeforeModify Driving Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <priority> InterruptedByModify None Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> AfterModify None Goal <goal> <robot>
<queuedDate> <queuedTime> None None 0
QueueUpdate: <id> <jobId> <modifiedPriority> Pending None Goal <goal> "None" <queuedDate>
<queuedTime> None None 0
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

Details

The queueModifyLocal command allows modification of goal or priority values for job segments in these job types:

- Dropoffs
- Swaps

It allows modification of segments in these states:

- Pending job segments
- InProgress jobs up to and including "InProgress Driving", but not after

Changing the priority for the first segment in a job may change the order in which it gets assigned. Changing the priority of other segments in the job will never change the order in which the job is assigned.

The queue time for a job will never be changed as a result of a queueModify command.

Changing the shared goal in a swap will break the link between the two jobs. Changing the other goals in the swap will not break the link.

Modified jobs will be candidates for swaps. The linking would occur immediately following the modify.

Examples

Example #1 – goal modify of a pending item:

```
queuePickup t
queuePickup goal "t" with priority 10 id PICKUP5 and jobId JOB5 successfully queued
QueueUpdate: PICKUP5 JOB5 10 Pending None Goal "t" "None" 03/25/2015 07:36:58 None None 0
queuemodify pickup5 goal w20
queuemodify modifying id pickup5 goal "w20"
QueueUpdate: PICKUP5 JOB5 10 BeforeModify None Goal "t" "None" 03/25/2015 07:36:58 None None 0
QueueUpdate: PICKUP5 JOB5 10 AfterModify None Goal "w20" "None" 03/25/2015 07:36:58 None None 0
QueueUpdate: PICKUP5 JOB5 10 Pending None Goal "w20" "None" 03/25/2015 07:36:58 None None 0

queueDropoff y
queueDropoff attempting to queue goal "y" using default priority
queueDropoff goal "y" with priority 20 id DROPOFF6 and jobId JOB6 successfully queued
QueueUpdate: DROPOFF6 JOB6 20 Pending None Goal "y" "robotOne" 03/25/2015 07:38:09 None None 0
queuemodifylocal dropoff6 goal x
queuemodifylocal modifying id dropoff6 goal "x"
QueueUpdate: DROPOFF6 JOB6 20 BeforeModify None Goal "y" "robotOne" 03/25/2015 07:38:09 None None 0
QueueUpdate: DROPOFF6 JOB6 20 AfterModify None Goal "x" "robotOne" 03/25/2015 07:38:09 None None 0
QueueUpdate: DROPOFF6 JOB6 20 Pending None Goal "x" "robotOne" 03/25/2015 07:38:09 None None 0
```

Example #2 – priority modify of a pending item:

```
queueDropoff w20
queueDropoff attempting to queue goal "w20" using default priority
queueDropoff goal "w20" with priority 20 id DROPOFF7 and jobId JOB7 successfully queued
QueueUpdate: DROPOFF7 JOB7 20 Pending None Goal "w20" "robotOne" 03/25/2015 07:39:01 None None 0
queuemodifylocal dropoff7 priority 22
queuemodifylocal modifying id dropoff7 priority 22
QueueUpdate: DROPOFF7 JOB7 20 BeforeModify None Goal "w20" "robotOne" 03/25/2015 07:39:01 None None 0
QueueUpdate: DROPOFF7 JOB7 22 AfterModify None Goal "w20" "robotOne" 03/25/2015 07:39:01 None None 0
QueueUpdate: DROPOFF7 JOB7 22 Pending None Goal "w20" "robotOne" 03/25/2015 07:39:01 None None 0

queuePickup v
queuePickup goal "v" with priority 10 id PICKUP8 and jobId JOB8 successfully queued
```

```
QueueUpdate: PICKUP8 JOB8 10 Pending None Goal "v" "None" 03/25/2015 07:40:24 None None 0
queuemodify pickup8 priority 6
queuemodify modifying id pickup8 priority 6
QueueUpdate: PICKUP8 JOB8 10 BeforeModify None Goal "v" "None" 03/25/2015 07:40:24 None None 0
QueueUpdate: PICKUP8 JOB8 6 AfterModify None Goal "v" "None" 03/25/2015 07:40:24 None None 0
QueueUpdate: PICKUP8 JOB8 6 Pending None Goal "v" "None" 03/25/2015 07:40:24 None None 0
```

Example #3 – goal modify of an inProgress item:

```
queuePickup x
queuePickup goal "x" with priority 10 id PICKUP9 and jobId JOB9 successfully queued
QueueUpdate: PICKUP9 JOB9 10 Pending None Goal "x" "None" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress UnAllocated Goal "x" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Allocated Goal "x" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Driving Goal "x" "robotTwo" 03/25/2015 07:47:21 None None 0
queuemodify pickup9 goal y
queuemodify modifying id pickup9 goal "y"
QueueUpdate: PICKUP9 JOB9 10 BeforeModify Driving Goal "x" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InterruptedByModify None Goal "x" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 AfterModify None Goal "y" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 Pending None Goal "y" "None" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress UnAllocated Goal "y" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Allocated Goal "y" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 InProgress Driving Goal "y" "robotTwo" 03/25/2015 07:47:21 None None 0
QueueUpdate: PICKUP9 JOB9 10 Completed None Goal "y" "robotTwo" 03/25/2015 07:47:21 03/25/2015 07:48:00 0
```

Example #4 – priority modify of an inProgress item:

```
queuePickup t
queuePickup goal "t" with priority 10 id PICKUP10 and jobId JOB10 successfully queued
QueueUpdate: PICKUP10 JOB10 10 Pending None Goal "t" "None" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 10 InProgress UnAllocated Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 10 InProgress Allocated Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 10 InProgress Driving Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
queuemodify pickup10 priority 13
queuemodify modifying id pickup10 priority 13
QueueUpdate: PICKUP10 JOB10 10 BeforeModify Driving Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 10 InterruptedByModify None Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 AfterModify None Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 Pending None Goal "t" "None" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 InProgress UnAllocated Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 InProgress Allocated Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 InProgress Driving Goal "t" "robotTwo" 03/25/2015 07:49:34 None None 0
QueueUpdate: PICKUP10 JOB10 13 Completed None Goal "t" "robotTwo" 03/25/2015 07:49:34 03/25/2015 07:49:46 0
```

Related Commands

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueMulti Command (shortcut: qm) on page 229

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queryFaults Command (shortcut: qf) on page 204

queueMulti Command (shortcut: qm)

Queues the robot for multiple pickups and dropoffs at multiple goals.

Syntax

queueMulti <number of goals> <number of fields per goal> <goal1> <goal1 args> <goal2> <goal2 args> … <goalN> <goalN args> [jobid]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

ARAM Settings

In order to use this feature, you have to explicitly enable it in the MobilePlanner software, by setting the EnterpriseQueuing argument in the Enterprise Features section of the **Configuration > Enterprise** tab.

Parameters

The queueMulti arguments are described in the table below.

For details on the data types, see Data Types on page 48.

Parameter	Definition
number of goals	Enter the number of goals where you want the mobile robot to go. Up to 50 goals are supported.
number of fields per goal	Enter the number of fields to be used for all goals. Two fields are supported, in this order: <pickup|dropoff> <priority>.
goal1	Enter the name of the first goal.
goal1 args	Enter the arguments associated with the first goal in the form: <pickup|dropoff> <priority or "default"> The first goal MUST be a pickup. All subsequent goals can be either pickups or dropoffs. The priority is an integer value that represents the priority of the job segment. The higher the number, the sooner the Enterprise Manager is going to service the item. The default priority is 10, which can be changed in MobilePlanner. Only the priority of the first segment in the queueMulti command will have an impact on how soon the job is assigned to a robot.
goalN	Enter the name of the Nth goal.
goalN args	Enter the arguments associated with the Nth goal.
jobId	Enter an optional identifier for the specified job. You can use a combination of string characters and integers. The jobId is helpful in tracking the job. If nothing is entered, ARCL generates a random jobId.

Responses

The command returns:

```
QueueMulti: goal "x" with priority 10 id PICKUP1 and jobid JOB1 successfully queued
QueueMulti: goal <"goal1"> with priority <goal1_priority> id <PICKUPid_or_DROPOFFid>
jobid <jobId> successfully queued
QueueMulti: goal <"goal2"> with priority <goal2_priority> id <PICKUPid_or_DROPOFFid>
jobid <jobId> successfully queued and linked to <goal1_PICKUPid_or_DROPOFFid>
:
:
QueueMulti: goal <"goaln"> with priority <goaln_priority> id <PICKUPid_or_DROPOFFid>
jobid <jobId> successfully queued and linked to <goal(n-1)_PICKUPid_or_DROPOFFid>
EndQueueMulti
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

Details

The queueMulti command tells the mobile robot to go to multiple goals, to make pickups and dropoffs.

Examples

The following example shows a queuedropoff at goal 1.

```
Example #1 - Using Default job id
queuemulti 4 2 x pickup 10 y pickup 19 z dropoff 20 t dropoff 20
QueueMulti: goal "x" with priority 10 id PICKUP1 and jobid JOB1 successfully queued
QueueMulti: goal "y" with priority 19 id PICKUP2 and jobid JOB1 successfully queued and
linked to PICKUP1
QueueMulti: goal "z" with priority 20 id DROPOFF3 and jobid JOB1 successfully queued and
linked to PICKUP2
QueueMulti: goal "t" with priority 20 id DROPOFF4 and jobid JOB1 successfully queued and
linked to DROPOFF3
EndQueueMulti
QueueUpdate: PICKUP1 JOB1 10 Pending None Goal "x" "None" 08/15/2013 06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 Pending ID_PICKUP1 Goal "y" "None" 08/15/2013 06:02:59 None
None 0
QueueUpdate: DROPOFF3 JOB1 20 Pending ID_PICKUP2 Goal "z" "None" 08/15/2013 06:02:59 None
None 0
QueueUpdate: DROPOFF4 JOB1 20 Pending ID_DROPOFF3 Goal "t" "None" 08/15/2013 06:02:59
None None 0
QueueUpdate: PICKUP1 JOB1 10 InProgress UnAllocated Goal "x" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: PICKUP1 JOB1 10 InProgress Allocated Goal "x" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP1 JOB1 10 InProgress Driving Goal "x" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP1 JOB1 10 Completed None Goal "x" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:03:20 0
QueueUpdate: PICKUP2 JOB1 19 InProgress UnAllocated Goal "y" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 InProgress Allocated Goal "y" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 InProgress Driving Goal "y" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 Completed None Goal "y" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:03:33 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress UnAllocated Goal "z" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress Allocated Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress Before Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress Driving Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress After Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 Completed None Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:03:47 0
QueueUpdate: DROPOFF4 JOB1 20 InProgress UnAllocated Goal "t" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
```

```
QueueUpdate: DROPOFF4 JOB1 20 InProgress Allocated Goal "t" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: DROPOFF4 JOB1 20 InProgress Driving Goal "t" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF4 JOB1 20 Completed None Goal "t" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:04:03 0
```

Related Commands

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queuePickup Command (shortcut: qp)

Calls any available robot for a pick up request.

Syntax

queuePickup <goalName> [priority or "default"] [jobId]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

ARAM Settings

In order to use this feature, you have to explicitly enable it in the MobilePlanner software, by setting the EnterpriseQueuing argument in the Enterprise Features section of the **Configuration > Enterprise** tab.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
goalName	Enter the name of the goal where you want the mobile robot to go for the pickup.
priority	An optional integer value that represents the priority of the pickup request. The higher the number, the sooner Enterprise Manager is going to service the item. The default priority is 10, which can be changed in MobilePlanner.
jobId	An optional identifier for the specified job. You can use a combination of string characters and integers. The jobId is helpful in tracking the job. If nothing is entered, ARCL generates a random jobId.

Responses

The command returns the following information:

```
queuepickup goal "goalName" with priority [priority] id (id) and jobId [jobid] suc-
cessfully queued
```

Assuming the command was successful, the status of the robot is displayed:

```
QueueUpdate: <id> <jobId> <priority> <status = Pending> <substatus = None> Goal
<”goalName”> <assigned robotName = None>  <queuedDate>  <queuedTime> <completedDate =
None> <completedTime = None> <failedCount>
```

```
QueueUpdate: <id> <jobId> <priority> <status = InProgress> <substatus = None> Goal
<”goalName”> <”robotName”>  <queuedDate>  <queuedTime> <completedDate = None> <com-
pletedTime = None> <failedCount>
QueueUpdate: <id> <jobId> <priority> <status = Completed> <substatus = None> Goal
<”goalName”> <”robotName”>  <queuedDate> <queuedTime> <completedDate> <completedTime>
<failedCount>
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

Details

The queuePickup command calls any available robot for a pick up request. When the job is at the top of the queue, the mobile robot drives to the specified goal.

If multiple robots are available for the pickup request, the Enterprise Manager determines which robot answers the request based on such factors as which robot is closest to the goal, how long it has been idle, and its charge state. You can also enter a priority value: the higher the number, the higher the priority.

Examples

The following example shows a queuePickup at goal z with priority 11 and job_id xyz.

```
queuepickup z 11 xyz
queuepickup goal "z" with priority 11, id PICKUP13 and job_id xyz successfully queued
QueueUpdate: PICKUP13 xyz 11 Pending None Goal "z" none 12/19/2011 06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress UnAllocated  Goal "z" “Adept_Telepresence_Robot”
12/19/2011 06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress Allocated  Goal "z" “Adept_Telepresence_Robot”
12/19/2011 06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress BeforePickup Goal "z" “Adept_Telepresence_Robot”
12/19/2011  06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress BeforeEvery Goal "z" “Adept_Telepresence_Robot”
12/19/2011  06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress Before Goal "z" “Adept_Telepresence_Robot”
12/19/2011  06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress Driving Goal "z" “Adept_Telepresence_Robot”
12/19/2011  06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress After Goal "z" “Adept_Telepresence_Robot”
12/19/2011  06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress AfterEvery Goal "z" “Adept_Telepresence_Robot”
12/19/2011  06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 InProgress AfterPickup  Goal "z" “Adept_Telepresence_Robot”
12/19/2011 06:54:18 None None 0
QueueUpdate: PICKUP13 xyz 11 Completed None Goal "z" “Adept_Telepresence_Robot”
12/19/2011  06:54:18 12/19/2011  06:54:34 0
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queuePickupDropoff Command (shortcut: qpd)

Queues a pick-up and drop-off request for any available robot.

Syntax

queuePickupDropoff <goal1Name> <goal2Name> [priority1 or "default"] [priority2 or "default"] [jobId]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
goal1Name	Enter the name of the goal where you want the mobile robot to go for the pickup.
goal2Name	Enter the name of the goal where you want the mobile robot to go for the dropoff.
priority1	An optional integer value that represents the priority of the pickup request. The higher the number, the sooner Enterprise Manager is going to service the item. The default priority is 10, which can be changed in MobilePlanner.
priority2	An optional integer value that represents the priority of the dropoff request. The higher the number, the sooner Enterprise Manager is going to service the item. The default priority is 20, which can be changed in MobilePlanner.
jobId	An optional identifier for the specified job. You can use a combination of string characters and integers. The jobId is helpful in tracking the job. If nothing is entered, ARCL generates a random jobId.

Responses

The command returns the following information:

```
queuepickupdropoff goals <"goal1"> and <"goal2"> with priorities <priority1> and <pri-
ority2> ids <PICKUPid> and <DROPOFFid> jobId <jobId> successfully queued and linked to
jobId <jobid>
```

The PICKUPid and DROPOFFid are assigned by the system.

Assuming the command was successful, the status is displayed as follows:

```
QueueUpdate: <id> <jobId> <priority> <status=Pending> <substatus=None> Goal <"goal1">
<robotName>  <queued date> <queued time> <completed date=None> <completed time=None>
<failed count>
QueueUpdate: <id> <jobId> <priority> <status=Pending> <substatus=ID_<id>> Goal <"goal2">
<robotName>  <queued date> <queued time> <completed date=None> <completed time=None>
<failed count>

QueueUpdate: <id> <jobId> <priority> <status=InProgress> <substatus=UnAllocated> Goal
<"goal1">  <robotName>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>
QueueUpdate: <id> <jobId> <priority> <status=InProgress> <substatus=Allocated> Goal
<"goal1">  <robotName>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>
QueueUpdate: <id> <jobId> <priority> <status=InProgress> <substatus=Driving> Goal
<"goal1">  <robotName>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>
QueueUpdate: <id> <jobId> <priority> <status=Completed> <substatus=None> Goal <"goal1">
<robotName>  <queued date> <queued time> <completed date> <completed time> <failed count>
QueueUpdate: <id> <jobId> <priority> <status=InProgress> <substatus=UnAllocated> Goal
<"goal2">  <robotName>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>
QueueUpdate: <id> <jobId> <priority> <status=InProgress> <substatus=Allocated> Goal
<"goal2">  <robotName>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>
QueueUpdate: <id> <jobId> <priority> <status=InProgress> <substatus=Driving> Goal
<"goal2">  <robotName>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>
QueueUpdate: <id> <jobId> <priority> <status=Completed> <substatus=None> Goal <"goal2">
<robotName>  <queued date> <queued time> <completed date> <completed time> <failed count>
```

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

Details

The queuePickupDropoff command calls any available robot for a pick-up request and then tells it to go to a specific goal for a dropoff. You must specify the goal names. You can optionally specify the priorities for each goal and the job identifier. However, note that there is no robot specification parameter in this command—it automatically chooses the most appropriate robot in the fleet, as determined by the selection criteria and task requirements.

Examples

The following example shows the queuepickupdropoff command with priority1 and priority2 values and a job identifier.

```
queuepickupdropoff  <PICKUPgoal_name> <DROPOFFgoal_name> [PICKUPpriority] [DROPOFFpri-
ority] [job_id]

Returns:

queuepickupdropoff goals <"PICKUPgoal"> and <"DROPOFFgoal"> with priorities <PICKUPpri-
ority> and <DROPOFFpriority> ids <PICKUPid> and <DROPOFFid> job_id <jobid> successfully
queued

QueueUpdate: <id> <job_id> <priority> <status=Pending> <substatus=None> Goal <”goal_
name”>  <robot_name>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>

QueueUpdate: <id> <job_id> <priority> <status=InProgress> <substatus=None> Goal <”goal_
name”>  <robot_name>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>

QueueUpdate: <id> <job_id> <priority> <status=Completed> <substatus=None> Goal <”goal_
name”>  <robot_name>  <queued date> <queued time> <completed date> <completed time>
<failed count>

QueueUpdate: <id> <job_id> <priority> <status=InProgress> <substatus=None> Goal <”goal_
name”>  <robot_name>  <queued date> <queued time> <completed date=None> <completed time-
e=None> <failed count>

QueueUpdate: <id> <job_id> <priority> <status=Completed> <substatus=None> Goal <”goal_
name”>  <robot_name>  <queued date> <queued time> <completed date> <completed time>
<failed count>
```

The following example shows the queuepickupdropoff command being used to swap the payload on the robot:

```
queuepickupdropoff x y
queuepickupdropoff goals "x" and "y" with priorities 10 and 20 ids PICKUP12 and
DROPOFF13 job_id JOB12 successfully queued
QueueUpdate: PICKUP12 JOB12 10 Pending None Goal "x" "None" 08/16/2012 14:32:54 None
None 0
QueueUpdate: DROPOFF13 JOB12 20 Pending None Goal "y" "None" 08/16/2012 14:32:54 None
None 0
QueueUpdate: PICKUP12 JOB12 10 InProgress UnAllocated Goal "x" "Lynx1" 08/16/2012
14:32:54 None None 0
queuepickupdropoff y t
queuepickupdropoff goals "y" and "t" with priorities 10 and 20 ids PICKUP14 and
DROPOFF15 job_id JOB14 successfully queued and linked to job_id JOB12
QueueUpdate: PICKUP14 JOB14 10 Pending None Goal "y" "Lynx1" 08/16/2012 14:33:01 None
None 0
QueueUpdate: DROPOFF15 JOB14 20 Pending None Goal "t" "Lynx1" 08/16/2012 14:33:01 None
None 0
QueueUpdate: PICKUP12 JOB12 10 InProgress Allocated Goal "x" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: PICKUP12 JOB12 10 InProgress Driving Goal "x" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: PICKUP12 JOB12 10 Completed None Goal "x" "Lynx1" 08/16/2012 14:32:54
```

```
08/16/2012 14:33:15 0
QueueUpdate: DROPOFF13 JOB12 20 InProgress UnAllocated Goal "y" "Lynx1" 08/16/2012
14:32:54 None None 0
QueueUpdate: DROPOFF13 JOB12 20 InProgress Allocated Goal "y" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: DROPOFF13 JOB12 20 InProgress Driving Goal "y" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: DROPOFF13 JOB12 20 Completed None Goal "y" "Lynx1" 08/16/2012 14:32:54
08/16/2012 14:33:27 0
QueueUpdate: PICKUP14 JOB14 10 Completed None Goal "y" "Lynx1" 08/16/2012 14:33:01
08/16/2012 14:33:27 0
QueueUpdate: DROPOFF15 JOB14 20 InProgress UnAllocated Goal "t" "Lynx1" 08/16/2012
14:33:01 None None 0
QueueUpdate: DROPOFF15 JOB14 20 InProgress Allocated Goal "t" "Lynx1" 08/16/2012 14:33:01
None None 0
QueueUpdate: DROPOFF15 JOB14 20 InProgress Driving Goal "t" "Lynx1" 08/16/2012 14:33:01
None None 0
QueueUpdate: DROPOFF15 JOB14 20 Completed None Goal "t" "Lynx1" 08/16/2012 14:33:01
08/16/2012 14:33:35 0
```

```
queuepickupdropoff x y
queuepickupdropoff goals "x" and "y" with priorities 10 and 20 ids PICKUP12 and DROPOFF13
job_id JOB12 successfully queued
QueueUpdate: PICKUP12 JOB12 10 Pending None Goal "x" "None" 08/16/2012 14:32:54 None None
0
QueueUpdate: DROPOFF13 JOB12 20 Pending ID_PICKUP12 Goal "y" "None" 08/16/2012 14:32:54
None None 0
QueueUpdate: PICKUP12 JOB12 10 InProgress UnAllocated Goal "x" "Lynx1" 08/16/2012
14:32:54 None None 0
queuepickupdropoff y t
queuepickupdropoff goals "y" and "t" with priorities 10 and 20 ids PICKUP14 and DROPOFF15
job_id JOB14 successfully queued and linked to job_id JOB12
QueueUpdate: PICKUP14 JOB14 10 Pending ID_DROPOFF13 Goal "y" "Lynx1" 08/16/2012 14:33:01
None None 0
QueueUpdate: DROPOFF15 JOB14 20 Pending ID_PICKUP14 Goal "t" "Lynx1" 08/16/2012 14:33:01
None None 0
QueueUpdate: PICKUP12 JOB12 10 InProgress Allocated Goal "x" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: PICKUP12 JOB12 10 InProgress Driving Goal "x" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: PICKUP12 JOB12 10 Completed None Goal "x" "Lynx1" 08/16/2012 14:32:54
08/16/2012 14:33:15 0
QueueUpdate: DROPOFF13 JOB12 20 InProgress UnAllocated Goal "y" "Lynx1" 08/16/2012
14:32:54 None None 0
QueueUpdate: DROPOFF13 JOB12 20 InProgress Allocated Goal "y" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: DROPOFF13 JOB12 20 InProgress Driving Goal "y" "Lynx1" 08/16/2012 14:32:54
None None 0
QueueUpdate: DROPOFF13 JOB12 20 Completed None Goal "y" "Lynx1" 08/16/2012 14:32:54
08/16/2012 14:33:27 0
QueueUpdate: PICKUP14 JOB14 10 Completed None Goal "y" "Lynx1" 08/16/2012 14:33:01
08/16/2012 14:33:27 0
QueueUpdate: DROPOFF15 JOB14 20 InProgress UnAllocated Goal "t" "Lynx1" 08/16/2012
14:33:01 None None 0
QueueUpdate: DROPOFF15 JOB14 20 InProgress Allocated Goal "t" "Lynx1" 08/16/2012 14:33:01
```

```
None None 0
QueueUpdate: DROPOFF15 JOB14 20 InProgress Driving Goal "t" "Lynx1" 08/16/2012 14:33:01
None None 0
QueueUpdate: DROPOFF15 JOB14 20 Completed None Goal "t" "Lynx1" 08/16/2012 14:33:01
08/16/2012 14:33:35 0
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueQuery Command (shortcut: qq)

Shows the job status of the queue by type or value.

Items will be displayed by priority. If, for example, dropoff priority is 20 and pickup priority is 10, then dropoff items will be displayed first, followed by pickup items.

Syntax

queueQuery <type> <value> [echoString]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
type	Enter the type of job. Valid types are: • id = the pickup or dropoff identification • jobId = the job identification • robotName = the robot name • status = the item status.
value	Enter the value that corresponds with the type used: For id, enter the pickup or dropoff identification, for example: PICKUP2 For jobid, enter the job identification, for example: JOB2 For robotname, enter the robot name, for example: robot_34 For status, enter one of the following values: • inprogress = queries a job with an InProgress status. • pending = queries a job with a Pending status. • interrupted = queries a job with an Interrupted status. • completed = queries a job with a Completed status. • cancelled = queries a job with a Cancelled status. • failed = queries a job with a Failed status.
echoString	An optional string that is appended to each line of the results.

Responses

The command returns the following for a pending item:

```
QueueQuery: <id> <jobId> <priority> <status> <substatus> Goal <"goalName"> <robotName>
<queued date> <queued time> <completed date> <completed time> <echostring> <failed
count>
EndQueueQuery
```

The returned items will be displayed by priority, as shown in the Examples. If, for example, dropoff priority is 20 and pickup priority is 10, the dropoff items will be displayed before the pickup items.

Details

The queueQuery command is used to view the status of the job queue. The queue can be queried by type (such as the robot name or job identification) or by the job status.

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

An optional string can be specified, which will be appended to each line of the results.

For details on the status conditions, see Status Conditions on page 51.

Examples

The following example shows the status of the completed jobs in the queue.

```
queuequery status completed xyz
QueueQuery: DROPOFF18 y4rt 22 Completed None Goal "x" "MT-490" 12/19/2011 07:07:53
12/19/2011  07:08:07 xyz 0
QueueQuery: DROPOFF16 abc 20 Completed None Goal "x" "MT-490" 12/19/2011  07:06:00
12/19/2011  07:06:16 xyz 0
QueueQuery: DROPOFF17 JOB17 20 Completed None Goal "z" "MT-490" 12/19/2011  07:06:21
12/19/2011  07:06:35 xyz 0
QueueQuery: DROPOFF19 yyy 20 Completed None Goal "x" "MT-490" 12/19/2011  07:08:49
12/19/2011  07:08:49 xyz 0
QueueQuery: DROPOFF20 yyy 20 Completed None Goal "x" "MT-490" 12/19/2011  07:09:08
12/19/2011  07:09:09 xyz 1
QueueQuery: DROPOFF21 JOB21 20 Completed None Goal "x" "MT-490" 12/19/2011  07:09:33
12/19/2011  07:09:34 xyz 0
QueueQuery: PICKUP12 xyz 11 Completed None Goal "t" "MT-490" 12/19/2011  06:53:51
12/19/2011  06:54:02 xyz 5
QueueQuery: PICKUP13 xyz 11 Completed None Goal "z" "Adept_Telepresence_Robot"
12/19/2011  06:54:18 12/19/2011  06:54:34 xyz 0
EndQueueQuery
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209 queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueQueryLocal Command (shortcut: qql)

Shows the job status of the robot queue by type or value.

Items will be displayed by priority. If, for example, dropoff priority is 20 and pickup priority is 10, then dropoff items will be displayed first, followed by pickup items.

Syntax

queueQueryLocal <type> <value> [echoString]

Because the queueQueryLocal command is only available on the robot, it assumes it applies only to the items queued for that robot. This is a powerful difference (and feature) of the "local" version of the command. So, for example, a "queueQuerylLocal status inprogress" command would allow you to query, based on inprogress status, all jobs queued for that particular robot.

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
type	Enter the type of job. Valid types are: • id = the pickup or dropoff identification • jobId = the job identification • robotName = the robot name • status = the item status.
value	Enter the value that corresponds with the type used: For id, enter the pickup or dropoff identification, for example: PICKUP2 For jobid, enter the job identification, for example: JOB2 For robotname, enter the robot name, for example: robot_34 For status, enter one of the following values: • inprogress = queries a job with an InProgress status. • pending = queries a job with a Pending status. • interrupted = queries a job with an Interrupted status. • completed = queries a job with a Completed status. • cancelled = queries a job with a Cancelled status. • failed = queries a job with a Failed status.
echoString	An optional string that is appended to each line of the results.

Responses

The command returns the following for a pending item:

```
QueueQuery: <id> <jobId> <priority> <status> <substatus> Goal <"goalName"> <robotName>
<queued date> <queued time> <completed date> <completed time> <echostring> <failed count>
EndQueueQuery
```

The returned items will be displayed by priority, as shown in the Examples. If, for example, dropoff priority is 20 and pickup priority is 10, the dropoff items will be displayed before the pickup items.

Details

The queueQuery command is used to view the status of the job queue. The queue can be queried by type (such as the robot name or job identification) or by the job status.

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

An optional string can be specified, which will be appended to each line of the results.

For details on the status conditions, see Status Conditions on page 51.

Examples

The following example shows the status of the completed jobs in the queue.

```
queuequery status completed xyz
QueueQuery: DROPOFF18 y4rt 22 Completed None Goal "x" "MT-490" 12/19/2011 07:07:53
12/19/2011  07:08:07 xyz 0
QueueQuery: DROPOFF16 abc 20 Completed None Goal "x" "MT-490" 12/19/2011  07:06:00
12/19/2011  07:06:16 xyz 0
QueueQuery: DROPOFF17 JOB17 20 Completed None Goal "z" "MT-490" 12/19/2011  07:06:21
12/19/2011  07:06:35 xyz 0
QueueQuery: DROPOFF19 yyy 20 Completed None Goal "x" "MT-490" 12/19/2011  07:08:49
12/19/2011  07:08:49 xyz 0
QueueQuery: DROPOFF20 yyy 20 Completed None Goal "x" "MT-490" 12/19/2011  07:09:08
12/19/2011  07:09:09 xyz 1
QueueQuery: DROPOFF21 JOB21 20 Completed None Goal "x" "MT-490" 12/19/2011  07:09:33
12/19/2011  07:09:34 xyz 0
QueueQuery: PICKUP12 xyz 11 Completed None Goal "t" "MT-490" 12/19/2011  06:53:51
12/19/2011  06:54:02 xyz 5
QueueQuery: PICKUP13 xyz 11 Completed None Goal "z" "Adept_Telepresence_Robot" 12/19/2011
 06:54:18 12/19/2011  06:54:34 xyz 0
EndQueueQuery
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueShow Command (shortcut: qs)

Shows the status of the last 11 jobs in the queue, including any jobs assigned to the robots and the status of each job. Oldest jobs are displayed first.

Syntax

queueShow [echoString]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Shows all jobs and all robots. To look at a specific job, use queueQuery. To look at a specific robot, use queueShowRobot.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
echoString	An optional string that is appended to each line of the results.

Responses

The command returns the following information:

```
QueueRobot: <robotName> <robotStatus> <robotSubstatus> <echoString>
QueueShow: <id> <jobId> <priority> <status> <substatus> Goal <"goalName"> <”robotName”>
<queued date> <queued time> <completed date> <completed time> <echoString> <failed count>
EndQueueShow
```

Details

The queueShow command provides a listing of all robots connected to the Enterprise Manager, and all jobs in the queue including those that are pending, interrupted, or are currently assigned to the robots. You do not specify a robot with this command. Instead, it lists the information for all robots. If you wish to look at a specific robot, use the queueShowRobot command. For details, see the queueShowRobot Command (shortcut: qsr) on page 251. If you wish to look at a specific job, use the queueQuery command. For details, see the queueQuery Command (shortcut: qq) on page 241.

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

An optional string can be specified, which will be appended to each line of the results.

Examples

```
queueshow
QueueRobot: "21" InProgress Driving ""
QueueRobot: "22" Available Available ""
QueueRobot: "23" Available Available ""
QueueRobot: "24" Available Available ""
QueueRobot: "25" Available Available ""
QueueRobot: "26" Available Available ""
QueueShow: PICKUP3 JOB3 10 Completed None Goal "1" "21" 11/14/2012 11:49:23 11/14/2012
11:49:23 "" 0
QueueShow: PICKUP4 JOB4 10 InProgress Driving Goal "7" "21" 11/14/2012 11:49:34 None
None "" 0
EndQueueShow
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueShowCompleted Command (shortcut: qsc)

Shows the jobs in the queue with a status of Completed, oldest first.

Syntax

queueshowcompleted [echoString]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

Shows only jobs with a status of Completed. To look at a specific job, use queueQuery. To look at a specific robot, use queueShowRobot.

The configuration parameter maxNumberOfCompletedItems, which has a default of 100, limits the number of completed jobs that will be kept in the queue.

The configuration parameter DeleteCompletedItemsMinutes, which has a default of 60, determines how long completed jobs will be kept in the queue. Jobs older than this will be deleted from the queue, and cannot be viewed.

Either of these two parameters can limit the number of jobs in the queue that are available for viewing with the queueShowCompleted command.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
echoString	An optional string that is appended to each line of the results.

Returns

The command returns the following information:

```
QueueShow: <id> <jobId> <priority> <status> <substatus> Goal <"goalName"> <”robotName”>
<queued date> <queued time> <completed date> <completed time> <echoString> <failed count>
EndQueueShowCompleted
```

Details

The queueShowCompleted command provides a listing of the jobs in the queue that are Completed, oldest first. You do not specify a robot with this command. Instead, it lists the information for all robots. If you wish to look at a specific robot, use the queueShowRobot command. For details, see the queueShowRobot Command (shortcut: qsr) on page 251. If you wish to look at a specific job, use the queueQuery command. For details, see the queueQuery Command (shortcut: qq) on page 241.

The reported jobId was either provided as part of the request, or was autogenerated by the Enterprise Manager software.

All failed counts are based on the jobId.

For details on the status conditions, see Status Conditions on page 51.

An optional string can be specified, which will be appended to each line of the results.

Examples

```
queueshowcompleted

QueueShow: PICKUP19 JOB19 10 Completed None Goal "t" "Bullwinkle (.53)" 05/06/2013
05:55:33 05/06/2013 05:56:02 "" 0
QueueShow: PICKUP21 JOB21 10 Completed None Goal "t" "guiabot_2010_09_20" 05/06/2013
06:00:21 05/06/2013 06:00:42 "" 0
QueueShow: PICKUP22 JOB22 10 Completed None Goal "t" "Bullwinkle (.53)" 05/06/2013
06:00:32 05/06/2013 06:01:05 "" 0
QueueShow: PICKUP23 JOB23 10 Completed None Goal "t" "guiabot_2010_09_20" 05/06/2013
06:01:03 05/06/2013 06:01:23 "" 0
EndQueueShowCompleted
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr) on page 251

queueShowRobot Command (shortcut: qsr)

Shows the status and substatus of all robots (or, optionally, a specific robot) connected to the Enterprise Manager.

Syntax

queueShowRobot [robotName or "default"] [echoString]

Usage Considerations

This ARCL command is available only on the Enterprise Manager.

This command does not return any job information; to view the queue and job information, use the queueShow command from ARCL on the Enterprise Manager.

Parameters

The command parameters are described in the following table.

For details on the data types, see Data Types on page 48.

Parameter	Definition
robotName	Enter the name of the robot. To view all the robots connected to the Enterprise Manager, omit this parameter or enter "default".
echoString	An optional string that is appended to each line of the results. Requires a value in the previous parameter.

Responses

The command returns the following:

```
QueueRobot: "robotName" robotStatus robotSubstatus echoString
EndQueueShowRobot
```

For details on the status conditions, see Status Conditions on page 51.

Details

The queueShowRobot command displays the status of the robots currently connected to the Enterprise Manager. Optionally, this command allows you to query a specific robot name, versus the queueShow command, which returns the queue status for all robots along with queue information.

This command does not return the job status for jobs currently in progress. To view that information, use the queueShow command. For details, see queueShow Command (shortcut: qs) on page 247.

An optional string can be specified, which will be appended to each line of the results.

Examples

The following example shows the status and substatus of robot 31:

```
queueshowrobot 31
QueueRobot: "31" Available Available ""
```

The following example shows the status and substatus of all robots and includes an optional message "echoit":

```
Queueshowrobot default echoit

QueueRobot: “Robot1” UnAvailable EStopPressed echoit
QueueRobot: “Robot2” UnAvailable Interrupted echoit
QueueRobot: “Robot3” UnAvailable InterruptedButNotYetIdle echoit
QueueRobot: “Robot4” Available Available echoit
QueueRobot: “Robot5” InProgress Driving  echoit
QueueRobot: “Robot6” UnAvailable NotUsingEnterpriseManager echoit
QueueRobot: “Robot7” UnAvailable UnknownBatteryType echoit
QueueRobot: “Robot8” UnAvailable ForcedDocked echoit
QueueRobot: “Robot9” UnAvailable NotLocalized echoit
QueueRobot: "patrolbot" UnAvailable Fault_Driving_Application_faultName echoit

EndQueueShowRobot
```

Related Commands

queryFaults Command (shortcut: qf) on page 204

queueCancel Command (shortcut: qc) on page 209

queueCancel Command (shortcut: qc) on page 209

queueDropoff Command (shortcut: qd) on page 215

queueMulti Command (shortcut: qm) on page 229

queuePickup Command (shortcut: qp) on page 233

queuePickupDropoff Command (shortcut: qpd) on page 236

queueQuery Command (shortcut: qq) on page 241

queueQuery Command (shortcut: qq) on page 241

queueShow Command (shortcut: qs) on page 247

queueShowCompleted Command (shortcut: qsc) on page 249

queueShowRobotLocal Command (shortcut: qsrl)

The queueShowRobotLocal command displays the status of the robot.

Syntax

queueshowrobotlocal [echo_string]

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The queueShowRobotLocal arguments are described in the table below.

Parameter	Definition
[echo_string]	Enter an optional string value that will be displayed at the end of the command.

Details

ARCL displays the following:

QueueRobot: robot_name robot_status robot_substatus echostring

Examples

The following example shows the queueShowRobotLocal command used to display the status and sub-status of the robot. It includes an optional message "echoit".

```
queueshowrobotlocal echoit
QueueRobot: “Robot1” UnAvailable EStopPressed echoit
EndQueueShowRobot
```

Related Commands

queueCancel Command

queueDropoff command

queuePickup command

queuePickupDropoff command

queueQuery Command

queueShow command

queueShowRobot Command

quit Command

Closes the connection to the server.

Syntax

rangeDeviceList

Usage Considerations

This ARCL command is available on the robot and Enterprise Manager.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Closing connection
```

Details

The quit command closes the ARCL client-server connection. It only closes the connection; it does not shut down the server. To do that, use the shutDownServer command. For details, see shutDownServer Command.

Examples

```
quit
```

The command returns:

```
Closing connection
```

Related Commands

shutdown Command on page 275

shutDownServer Command

stop Command on page 278

rangeDeviceGetCumulative Command

Gets the cumulative readings of a range device.

Syntax

rangeDeviceGetCumulative <name>

Usage Considerations

This ARCL command is only available on the robot.

This parameter is case-sensitive.

The robot may not sense anything if operated in an open area, and then it would not provide cumulative readings.

ARAM Settings

For custom sensors: This command requires the addition of the "-customSensor <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name for the device. This parameter is case-sensitive.

Responses

The command returns:

```
RangeDeviceGetCumulative: <name> <series of X and Y points>
```

Details

The rangeDeviceGetCumulative command returns persistent readings from the named ranging sensor. This parameter is case-sensitive.

The X-Y coordinates are map points (in mm) for center of a polyDot or tip of a polyArrow, which display-aligns toward the center of the robot.

Note that not all sensors provide cumulative readings. For example, custom sensors do not provide cumulative readings. Lasers are the main sensors that provide cumulative readings that are useful. You can use the rangeDeviceList command to identify the sensors that provide cumulative readings. If there is no

<rangeDeviceName>CumulativeDrawingData listed for a sensor, then that sensor doesn't provide cumulative readings.

The robot may not sense anything if operated in an open area, and then it would not provide cumulative readings.

Examples

The following command returns the cumulative readings for the device "sim_lms2xx_1":

```
rangedevicegetcumulative sim_lms2xx_1
```

The command returns:

```
RangeDeviceGetCumulative: sim_lms2xx_1  -15600 10684  -15016 10609  -14464 10603
            -14712 10921  -14177 10706  -11343 10444  -14834 10722  -12696 10659
            -12485 10621  -12056 10621  -11838 10643  -11630 10706  -10364 10667
            -14471 10944  -10820 11366  -11315 10683  -13950 10689  -13739 10693
            -13525 10692  -13309 10697  -12256 10622  -12898 10701  -13103 10700
            -15254 10706  -15048 10937  -15041 11165  -15162 11555
```

Related Commands

customReadingAdd Command on page 96

customReadingAddAbsolute Command on page 94

customReadingsClear Command on page 98

rangeDeviceGetCumulative Command on page 255

rangeDeviceGetCurrent Command on page 257

rangeDeviceList Command on page 259

rangeDeviceGetCurrent Command

Gets the current readings of a range device.

Syntax

rangeDeviceGetCurrent <name>

Usage Considerations

This ARCL command is only available on the robot.

This parameter is case-sensitive.

ARAM Settings

For custom sensors: This command requires the addition of the "-customSensor <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that represents the name for the device. This parameter is case-sensitive.

Responses

The command returns:

```
RangeDeviceGetCurrent: <name> <series of X and Y points>
```

Details

The rangeDeviceGetCurrent command returns only active (current) detection readings from the named ranging sensor. This parameter is case-sensitive.

To get the cumulative readings from a range device, use the rangeDeviceGetCumulative command. For details, see rangeDeviceGetCumulative Command on page 255.

The X-Y coordinates are map points (in mm) for the center of a polyDot or tip of the polyArrow, which display-aligns toward the center of the robot.

Examples

The following command returns the current reading for the device "ARCL_CustomSensor":

```
rangedevicegetcurrent ARCL_CustomSensor
```

The command returns:

```
RangeDeviceGetCurrent: ARCL_CustomSensor  -2004 6921
```

Related Commands

customReadingAdd Command on page 96

customReadingAddAbsolute Command on page 94

customReadingsClear Command on page 98

rangeDeviceGetCumulative Command on page 255

rangeDeviceGetCurrent Command on page 257

rangeDeviceList Command on page 259

rangeDeviceList Command

Returns the list of range (sensor) devices.

Syntax

rangeDeviceList

Usage Considerations

This ARCL command is only available on the robot.

ARAM Settings

For custom sensors: This command requires the addition of the "-customSensor <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

This command does not have any parameters.

Responses

The command returns:

```
RangeDevice: <name> <type>
RangeDevice: <name> <icon> <RGB primary> <RGB secondary> <size mm> <layer> <defaultOn or
defaultOff>
RangeDeviceCumulativeDrawingData:” <name> <icon> <RGB primary> <RGB secondary> <size mm>
<layer> <defaultOn or defaultOff>
...
EndOfRangeDeviceList
```

Details

The rangeDeviceList command returns a list of the range (sensor) devices available in the current map. It also provides details about each device on the list. For details on the information returned, see the Responses section.

Examples

```
rangeDeviceList
```

The command returns:

```
RangeDevice: switchableForbidden LOCATION_DEPENDENT
RangeDeviceCurrentDrawingData: switchableForbidden polyDots 0xf9bd30 0x000000 35 73
DefaultOff
RangeDeviceCumulativeDrawingData: switchableForbidden polyDots 0x8b6305 0x0 50 60
DefaultOff
```

```
RangeDevice: Single_Robot_Sector LOCATION_DEPENDENT
RangeDeviceCurrentDrawingData: Single_Robot_Sector polyDots 0xd00000 0x000000 33 74
DefaultOff
RangeDevice: multiRobotCentral LOCATION_DEPENDENT
RangeDeviceCurrentDrawingData: multiRobotCentral polyDots 0x7d7d00 0x000000 100 72
DefaultOff
RangeDeviceCumulativeDrawingData: multiRobotCentral polyDots 0x7d007d 0x0 100 72
DefaultOff
RangeDevice: ARCL_GlobalCustomSensor NONE
RangeDeviceCurrentDrawingData: ARCL_GlobalCustomSensor polyDots 0xc5996c 0x000000 40 78
DefaultOn
RangeDevice: ARCL_CustomSensor NONE
RangeDeviceCurrentDrawingData: ARCL_CustomSensor polyDots 0xe9d324 0x000000 40 78
DefaultOn
RangeDevice: forbidden LOCATION_DEPENDENT
RangeDevice: irs NONE
RangeDeviceCurrentDrawingData: irs polyArrows 0xffff00 0x000000 120 80 DefaultOn
RangeDevice: bumpers NONE
RangeDeviceCurrentDrawingData: bumpers polyDots 0x000000 0x000000 120 83 DefaultOn
RangeDevice: sonar NONE
RangeDeviceCurrentDrawingData: sonar polyArrows 0x33ccff 0x000000 200 70 DefaultOn
RangeDevice: sim_lms2xx_1 LASER
RangeDeviceCurrentDrawingData: sim_lms2xx_1 polyDots 0x0000ff 0x000000 80 75 DefaultOn
RangeDeviceCumulativeDrawingData: sim_lms2xx_1 polyDots 0x7f 0x0 110 60 DefaultOn
EndOfRangeDeviceList
```

Related Commands

customReadingAdd Command on page 96

customReadingAddAbsolute Command on page 94

customReadingsClear Command on page 98

rangeDeviceGetCumulative Command on page 255

rangeDeviceGetCurrent Command on page 257

rangeDeviceList Command on page 259

say Command

Speak a text string through the robot audio output.

Syntax

say <text_string>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
string	Enter the text string that you want the mobile robot to say. Quotes are optional.

Responses

The command returns:

```
Saying <text_string>
```

Details

Allows you to have the mobile robot speak and then wait until it is finished before continuing on the route.

The say command is equivalent to the sayInstant task, which generates text-to-speech to the robot’s audio output, if enabled.

To have the robot play a sound (.wav) file, use the play command. For details, see play Command on page 199.

Examples

The following example commands the robot to say "hello":

```
say "hello"
Saying "hello"
```

Related Commands

play Command on page 199

scanAddGoal Command

Adds a goal while the robot is scanning.

Syntax

scanAddGoal <name> [description]

Usage Considerations

This ARCL command is only available on the robot.

The command can only be used while a scan is running.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that will represent the name for the goal.
description	Enter an optional description for the goal.

Responses

The command returns the new goal information:

```
mapAddGoal: Added goal with name <name> 'label'
```

Details

The scanAddGoal command adds a goal to the map while the robot is scanning. The goal is placed at the robot's current pose and includes the goal name and description. It is similar to using the goal button on the joystick, while scanning.

Examples

For example:

```
scanaddgoal goal_lc "This is Bob's goal"
```

The command returns:

```
mapAddGoal: Added goal with name 'goal_lc' and description 'This is Bob's goal'
```

Related Commands

scanAddInfo Command on page 264

scanAddTag Command on page 267

scanStart Command on page 269

scanStop Command on page 271

scanAddInfo Command

Adds an information line while the robot is scanning.

Syntax

scanaddinfo <LogInfo:*type*> <Name=*string*> <Label=*string*> <Desc=*string*> [Size =*integer*] [IsData=*integer*] [Vis=*mode*] [FtSize=*size*] [Color*n*=*value*] [Shape=*shape*]

Usage Considerations

This ARCL command is only available on the robot.

All parameter labels must be included, for example: LogInfo:GoalType Name=Newgoal Label="New Goal" Desc="This is a goal", etc.

The command can only be used while a scan is running.

All of fields and type names in the map file are case sensitive; if the case is wrong they are ignored. Unknown fields are ignored.

The objects are informational-only (they do not contain any position), versus those added by the scanAd-dTag command, which are position-corrected. For details, see scanAddTag Command on page 267.

ARAM Settings

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
LogInfo:*type*	Required. This defines the type of information you want to add. Valid *type* entries are: • GoalType • LocationType • SectorType GoalType markers are goals that can be driven to and appear in the map. LocationType and SectorType markers also appear in the map, but cannot be driven to. GoalType markers must have a unique name, LocationType and SectorType do not.
Name=*string*	Required. Enter a text string that represents the name of the type that is being defined.
Label=*string*	Required. Enter a text string that represents the label that is displayed for the type in popup menus, etc.
Desc=*string*	Required. Enter a text string that represents the description of the type that is displayed in tool tips, etc.
Size=*integer*	Enter an integer that represents the width/height (square) in mm of the marker's display icon in the map.
IsData=*integer*	Optional. Enter a 1 or 0. The default value is 0. Set IsData to 1 to have the related markers carried over with an 'insert map'.
Vis=*mode*	The default setting is DefaultOn. Valid modes are: • AlwaysOn • AlwaysOff • DefaultOn • DefaultOff Currently supported for IsData=1 markers only, specifies marker visibility. Go to the Map:Data menu in MobileEyes or MobilePlanner to change their visibility in the map display.
FtSize=*size*	Optional. Enter an integer that represents the point size for the marker's label and description font. If the font is scaled, then express the point size in mm and it must be a multiple of 100. If the font is fixed, express the point size in pixels.
Color*n*=*value*	Optional. For *n*, enter an integer from 0 - 2. For *value*, enter a color value which defines the marker's map-display icon colors. The color value is a six digit hexadecimal number (hence, the '0x' prefix). The first two digits give the red component, followed by two digits for green and two digits for blue. For example: • 0xFF0000 is bright red • 0x888888 is medium gray

Parameter	Definition
Shape=*shape*	Optional. Shape of the marker's map-display icon. The default setting is Plain. Valid shapes are: • Plain: The default shape, typically a filled square • Cross: A cross shape • Triangle: A triangle shape • Hbars: A square shape containing horizontal bars or stripes • Vbars: A square shape containing vertical bars or stripes • T: A "T" shape • U: A "U" shape

Responses

The command returns the new object information:

```
mapAddInfo: Added info 'LogInfo:type Name=string Label=string Desc=string Colorn=value
         IsData=integer Vis=mode FtSize=size Shape=shape'
```

Details

Adds an information line (a description, typically defining a custom object) to the beginning of the map for later reference.

This command only describes the object, it does not actually add any objects. That is done with the add ScanTag command. For details, see scanAddTag Command on page 267.

This command can only be used after a scan is started with the scanStart command. For details on starting a scan, see scanStart Command on page 269.

Examples

The following example adds a description of a custom LocationType object named "SensorReading":

```
scanAddInfo LogInfo:LocationType Name=SensorReading Label=”Sensor reading” Desc=”A
         sensor reading” Color=0x5500EE IsData=1 Vis=DefaultOn Shape=Cross
```

The command returns:

```
mapAddInfo: Added info 'LogInfo:LocationType Name=SensorReading Label= A sensor reading.
         Desc=.Blobs of paint. Color=0x5500EE IsData=1 Vis=DefaultOn Shape=Cross'
```

Related Commands

scanAddGoal Command on page 262

scanAddTag Command on page 267

scanStart Command on page 269

scanStop Command on page 271

scanAddTag Command

Adds a tag as a cairn (marker) while the robot is scanning.

Syntax

scanAddTag cairn:<name> [label] [icon_type] [description]

Usage Considerations

This ARCL command is only available on the robot.

The command can only be used while a scan is running.

The objects are position-corrected, versus those added by the scanAddInfo command, which contain no position information. For details, see scanAddInfo Command on page 264.

ARAM Settings

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter the name of the object, from the Name= parameter, created with the scanAddInfo command.
label	Enter an optional string that provides the icon label.
icon_type	Enter an optional string that provides the icon type.
description	Enter an optional string that provides a description of the marker.

Responses

The command returns the new object information:

```
mapAddTag 'cairn: <name>[WithHeading] "label" ICON "
```

Details

The scanAddTag command adds a marker in the map of type <name>, as described by a scanAddInfo of the same name. Objects are added to the scan as "cairns" (markers). The objects are position-corrected, versus those added by the scanAddInfo command, which contain no position information.

Add "WithHeading" as suffix to the name in order to include a heading in the marker's properties. For details, see the Examples section.

This command only adds a marker for the object, it does not create or describe the object. That is done with the add scanAddInfo command. For details, see scanAddInfo Command on page 264.

This command can only be used after a scan is started with the scanStart command. For details on starting a scan, see scanStart Command on page 269.

Examples

For example, suppose you used scanAddInfo to create an object description named "SensorReading", you can use it with the scanAddTag command, as follows:

To add objects of that type with a heading:

```
scanaddtag cairn: SensorReadingWithHeading "id1" ICON ""
```

and with and icon description:

```
scanaddtag cairn: RoomGoalWithHeading "Room 32920" ICON "Bob's office"
```

To add objects of that type without a heading:

```
scanaddtag cairn: SensorReading "id2" ICON ""
```

and with an icon description:

```
scanaddtag cairn: RoomGoal "Room 223203" ICON "Fred's office"
```

Related Commands

scanAddGoal Command on page 262

scanAddInfo Command on page 264

scanStart Command on page 269

scanStop Command on page 271

scanStart Command

Starts scanning mode with given 2d file name.

Syntax

scanStart <name>

Usage Considerations

This ARCL command is only available on the robot.

Only one scan can be running.

Parameters

The command parameters are described in the following table.

Parameters	Definition
name	Enter a string that will represent the name for the scan.

Responses

The command returns information:

```
Starting scan '<name>'
Scanning: Started scan '<name>'
```

Details

The scanStart command initiates scanning mode with the specified 2d file name. Only one scan can be active at a time. To stop the scan, use the scanStop command. For details, see scanStop Command on page 271.

Examples

To start a scan named "testscan", enter:

```
scanstart testscan
```

The command returns:

```
Starting scan 'testscan'
Scanning: Started scan 'testscan'
```

Related Commands

scanAddGoal Command on page 262

scanAddInfo Command on page 264

scanAddTag Command on page 267

scanStop Command on page 271

scanStop Command

Stops the started scan.

Syntax

scanStop

Usage Considerations

This ARCL command is only available on the robot.

The command can only be used while a scan is running.

Parameters

This command does not have any parameters.

Responses

The command returns information about the new piece of information in the following format:

```
Stopped scan '<name>'
Scanning: Stopped scan '<name>'
```

Examples

To stop the scan that's currently running, enter:

```
scanstop
```

The command returns:

```
Stopping scan 'testscan'
Scanning: Stopped scan 'testscan'
```

Related Commands

scanAddGoal Command on page 262

scanAddInfo Command on page 264

scanAddTag Command on page 267

scanStart Command on page 269

setPayload Command

Sets the payload name.

Syntax

setPayload <payload>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
payload	Enter a name for the payload; quotes are optional.

Responses

The command returns:

```
payload <payload>
```

Details

The setPayload command sets the name of the robot payload. The name can be more than one word; quotes are optional when entering the string. To view the payload name, use the getPayload command. For details, see getPayload Command on page 132.

If the robot has multiple slots (containers) with different payloads in each slot, use the payloadSet command to set the payload name for each slot. For details, see payloadSet Command (shortcut: ps) on page 194.

Examples

The following example sets the payload name as "This has widgets":

```
setpayload This has widgets
```

The command returns:

```
payload This has widgets
```

Related Commands

getPayload Command on page 132

payloadQuery Command (shortcut: pq) on page 187

payloadQuery Command (shortcut: pq) on page 187

payloadRemove Command (shortcut: pr) on page 192

payloadSet Command (shortcut: ps) on page 194

payloadSlotCount Command (shortcut: psc) on page 196

payloadSlotCountLocal Command (shortcut: pscl) on page 198

setPrecedence Command

Sets precedence for this robot in a multi-robot encounter. Lower values take higher precedence.

Syntax

setPrecedence <integer>

Usage Considerations

This ARCL command is only available on the robot.

Parameters

The command parameters are described in the following table.

Parameter	Definition
integer	Enter an integer value that determines the precedence of the robot in the event of a multi-robot encounter. The range is -100 to 100.

Responses

The command returns:

```
setPrecedence: <integer>
```

Details

The setPrecedence command is used to set the precedence information for the robot. The precedence value is used in a multi-robot encounter. The robot that has the lowest value will get highest precedence, the robot with the next lowest value will get the next highest precedence, and so on. The precedence value is viewed using the getPrecedence command. For details, see getPrecedence Command on page 133.

Examples

The following example sets the robot precedence to 20:

```
setprecedence 20
setPrecedence: 20
```

Related Commands

getPrecedence Command on page 133

shutdown Command

Shuts down the robot.

Syntax

shutDownServer

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

There is no response returned from this command.

Details

The shutdown command tells the robot to initiate its power-down sequence. The command works only with Adept mobile robots that have power-down hardware.

Examples

To shut down the robot, enter:

```
shutdown
```

There is no response returned from this command.

Related Commands

quit Command on page 254

shutDownServer Command

stop Command on page 278

status Command

Returns the operational state of the robot.

Syntax

status

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
Status: <status>
BatteryVoltage: <volts_dc>
Location: <X> <Y> <Theta>
LocalizationScore: <score>
Temperature: <degrees>
```

Details

The status command returns the operational state of the robot, such as docking or going to a goal, battery charge, position and localization score. To get a one-line status of the robot, use the oneLineStatus command. For details, see oneLineStatus Command on page 169.

Examples

To get the current status of the robot, enter the following:

```
status
```

The command returns:

```
Status: DockingState: Docking ForcedState: Unforced ChargeState: Not
BatteryVoltage: 26.1
Location: -969 301 1
LocalizationScore: 0.988304
Temperature: -128
```

Related Commands

getDateTime Command on page 123

getGoals Command on page 124

getInfo Command on page 126

getInfoList Command on page 128

getPayload Command on page 132

getRoutes Command on page 134

oneLineStatus Command on page 169

queryDockStatus Command on page 203

queryMotors Command on page 207

stop Command

Stops the current robot motion.

Syntax

stop

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
<status_message>
Stopping
Stopped
```

Examples

To stop the robot while it is moving to goal "g_25", enter the following:

```
stop
```

The command returns:

```
Interrupted: Going to g_25
Stopping
Stopped
```

Related Commands

quit Command on page 254

shutdown Command on page 275

shutDownServer Command

trackSectors Command

Returns all sectors at the robot's current position.

Syntax

trackSectors

Usage Considerations

This ARCL command is only available on the robot.

ARAM Settings

This command requires the addition of the "-trackSector <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

This command does not have any parameters.

Responses

The command returns:

```
TrackSectors: <type> [sector_name]
...
End of TrackSectors
```

Details

The trackSectors command lists all sectors at the robot's current position. If the robot is not on a sector, no information is returned. If a sector has no name, then only its type will be displayed.

Before using this command, you must first create sector types for the map file, which can be added and manipulated in MobilePlanner and visible in MobileEyes. To do this, you can download the map file, open it with a text editor (like Notepad) and add your own SectorType to the map file:

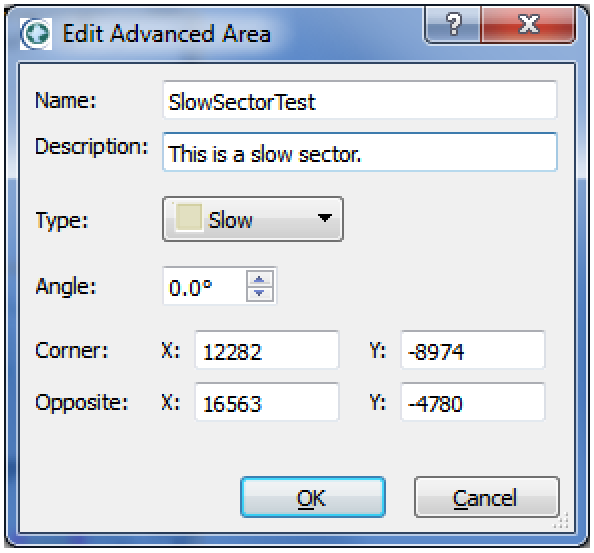
```
MapInfo: SectorType Name=SomeSector "Label=SomeLabel" "Desc=Some description" Shape=Plain
Color0=0x0088ff
```

For example, you might enter something like:

```
MapInfo: SectorType Name=VerySlowSector "Label=VerySlow" "Desc=Area in which the robot
drives very slowly" Shape=Plain Color0=0xffffb0
```

The entry must be on one line—it is easiest is just to copy/paste an existing sector and then change its parameters. There can be as many sector types as needed for your application. If you've given ARAM track sector arguments, it will make the tracking commands available over ARCL.

You can edit the sector properties from MobilePlanner by right-clicking on it and selecting Edit. The "Edit Advanced Area dialog opens, as shown in the following figure. You can add a name, description, select the type, and adjust the size and position.



Name: SlowSectorTest
Description: This is a slow sector.
Type: Slow
Angle: 0.0°
Corner: X: 12282 Y: -8974
Opposite: X: 16563 Y: -4780

Sector Properties

There are other "sector" commands, such as: 'trackSectorsAtGoal <goal>', which provides the sectors at a goal; 'trackSectorsAtPoint <X> <Y>', which provides the sectors at a point; and 'trackSectorsPath <optional:dist>', which provides the sectors for that distance along the path (or the whole path if no distance is given). Note that all of these commands have identical output, which is shown in the Responses section.

For details on the related commands, see the Related Commands section.

Examples

The following example lists the sectors at the robot's current position:

```
tracksectors
TrackSectors: SlowSector
End of TrackSectors
```

The following example shows the sector name after the sector in the previous example is named "SlowSectorTest" in MobilePlanner:

```
tracksectors
TrackSectors: SlowSector SlowSectorTest
End of TrackSectors
```

Related Commands

trackSectors Command on page 279

trackSectorsAtGoal Command on page 282

trackSectorsAtPoint Command on page 285

trackSectorsPath Command on page 288

trackSectorsAtGoal Command

List all sectors pertaining to a goal.

Syntax

trackSectorsAtGoal <goal>

Usage Considerations

This ARCL command is only available on the robot.

ARAM Settings

This command requires the addition of the "-trackSector <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
goal	Enter a string that represents the name for the goal.

Responses

The command returns:

```
TrackSectors: <type> [sector_name]
...
End of TrackSectors
```

Details

The trackSectors command lists all sectors at the named goal. If the named goal is not on a sector, no information is returned. If a sector has no name, then only its type will be displayed.

Before using this command, you must first create sector types for the map file, which can be added and manipulated in MobilePlanner and visible in MobileEyes. To do this, you can download the map file, open it with a text editor (like Notepad) and add your own SectorType to the map file:

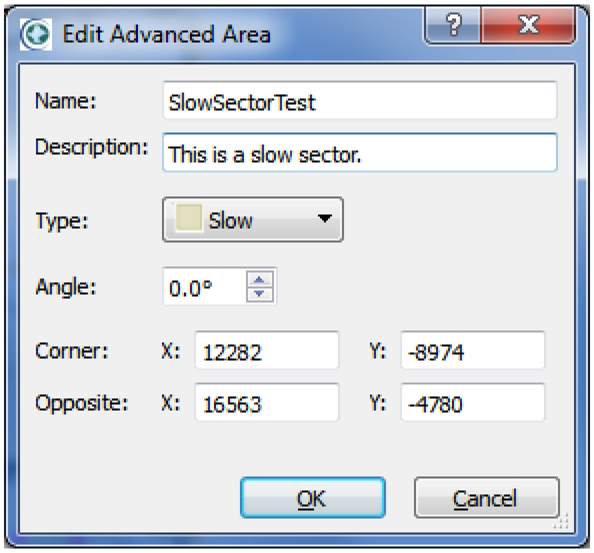
```
MapInfo: SectorType Name=SomeSector "Label=SomeLabel" "Desc=Some description" Shape-
e=Plain Color0=0x0088ff
```

For example, you might enter something like:

```
MapInfo: SectorType Name=VerySlowSector "Label=VerySlow" "Desc=Area in which the robot
drives very slowly" Shape=Plain Color0=0xffffb0
```

The entry must be on one line—it is easiest is just to copy/paste an existing sector and then change its parameters. There can be as many sector types as needed for your application. If you've given ARAM track sector arguments, it will make the tracking commands available over ARCL.

You can edit the sector properties from MobilePlanner by right-clicking on it and selecting Edit. The "Edit Advanced Area dialog opens, as shown in the following figure. You can add a name, description, select the type, and adjust the size and position.



Name: SlowSectorTest
Description: This is a slow sector.
Type: Slow
Angle: 0.0°
Corner: X: 12282 Y: -8974
Opposite: X: 16563 Y: -4780

Sector Properties

There are other "sector" commands, such as: 'trackSectors', which provides the sectors at the robot's current position; 'trackSectorsAtPoint <X> <Y>', which provides the sectors at a point; and 'trackSectorsPath <optional:dist>', which provides the sectors for that distance along the path (or the whole path if no distance is given). Note that all of these commands have identical output, which is shown in the Responses section.

For details on the related commands, see the Related Commands section.

Examples

The following example lists the sectors at the named goal:

```
tracksectorsatgoal g_10
TrackSectors: SlowSector
End of TrackSectors
```

The following example shows the sector name after the sector in the previous example is named "SlowSectorTest" in MobilePlanner:

```
tracksectorsatgoal g_10
TrackSectors: SlowSector SlowSectorTest
End of TrackSectors
```

Related Commands

trackSectors Command on page 279

trackSectorsAtPoint Command on page 285

trackSectorsPath Command on page 288

trackSectorsAtPoint Command

List all sectors pertaining to a point on the map.

Syntax

trackSectorsAtPoint <X> <Y>

Usage Considerations

This ARCL command is only available on the robot.

ARAM Settings

This command requires the addition of the "-trackSector <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
X	Enter the X coordinate for the point.
Y	Enter the Y coordinate for the point.

Responses

The command returns:

```
TrackSectors: <type> [sector_name]
...
End of TrackSectors
```

Details

The trackSectorsAtPoint command lists all sectors at the specified X-Y coordinates on the map. If the specified point is not on a sector, no information is returned. If a sector has no name, then only its type will be displayed.

Before using this command, you must first create sector types for the map file, which can be added and manipulated in MobilePlanner and visible in MobileEyes. To do this, you can download the map file, open it with a text editor (like Notepad) and add your own SectorType to the map file:

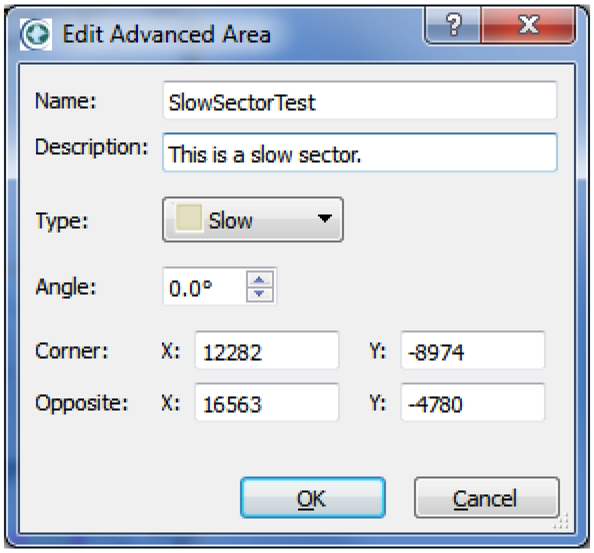
```
MapInfo: SectorType Name=SomeSector "Label=SomeLabel" "Desc=Some description" Shape=Plain
Color0=0x0088ff
```

For example, you might enter something like:

```
MapInfo: SectorType Name=VerySlowSector "Label=VerySlow" "Desc=Area in which the robot
drives very slowly" Shape=Plain Color0=0xffffb0
```

The entry must be on one line—it is easiest is just to copy/paste an existing sector and then change its parameters. There can be as many sector types as needed for your application. If you've given ARAM track sector arguments, it will make the tracking commands available over ARCL.

You can edit the sector properties from MobilePlanner by right-clicking on it and selecting Edit. The "Edit Advanced Area dialog opens, as shown in the following figure. You can add a name, description, select the type, and adjust the size and position.



Name: SlowSectorTest
Description: This is a slow sector.
Type: Slow
Angle: 0.0°
Corner: X: 12282 Y: -8974
Opposite: X: 16563 Y: -4780

Sector Properties

There are other "sector" commands, such as: 'trackSectors', which provides the sectors at the robot's current position; 'trackSectorsAtGoal <goal>', which provides the sectors at a point; and 'trackSectorsPath <optional:dist>', which provides the sectors for that distance along the path (or the whole path if no distance is given). Note that all of these commands have identical output, which is shown in the Responses section.

For details on the related commands, see the Related Commands section.

Examples

The following example lists the sectors at the specified coordinates:

```
tracksectorsatpoint 16615 -6497
TrackSectors: SlowSector
End of TrackSectors
```

The following example shows the sector name after the sector in the previous example is named "SlowSectorTest" in MobilePlanner:

```
tracksectorsatpoint 16615 -6497
TrackSectors: SlowSector SlowSectorTest
End of TrackSectors
```

Related Commands

trackSectors Command on page 279

trackSectorsAtGoal Command on page 282

trackSectorsPath Command on page 288

trackSectorsPath Command

Lists the tracked sectors the path (or part of it) is in. An optional distance can be specified.

Syntax

trackSectorsPath [distance]

Usage Considerations

This ARCL command is only available on the robot.

The robot must be traveling on a path when this command is issued; otherwise, no results will be returned.

ARAM Settings

This command requires the addition of the "-trackSector <name>" argument to the Custom Arguments section of the **Configuration > Debug** tab in the MobilePlanner software. For details, see the *Adept Motivity Software User's Guide.*

Parameters

The command parameters are described in the following table.

Parameters	Definition
distance	Enter an optional distance (in mm) from the front of the robot. Sectors will be listed that are on the path between the robot and that end point.

Responses

The command returns:

```
TrackSectors: <type> [sector_name]
...
End of TrackSectors
```

Details

The trackSectorsPath command lists all sectors in the path of the robot and, optionally, over a specified distance on its path. If the path (or part of it) is not on a sector, no information is returned. If a sector has no name, then only its type will be displayed.

Before using this command, you must first create sector types for the map file, which can be added and manipulated in MobilePlanner and visible in MobileEyes. To do this, you can download the map file, open it with a text editor (like Notepad) and add your own SectorType to the map file:

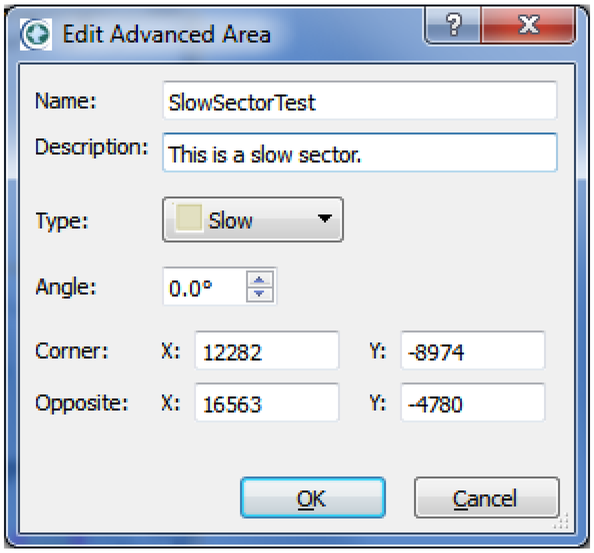
```
MapInfo: SectorType Name=SomeSector "Label=SomeLabel" "Desc=Some description" Shape-
e=Plain Color0=0x0088ff
```

For example, you might enter something like:

```
MapInfo: SectorType Name=VerySlowSector "Label=VerySlow" "Desc=Area in which the robot
drives very slowly" Shape=Plain Color0=0xffffb0
```

The entry must be on one line—it is easiest is just to copy/paste an existing sector and then change its parameters. There can be as many sector types as needed for your application. If you've given ARAM track sector arguments, it will make the tracking commands available over ARCL.

You can edit the sector properties from MobilePlanner by right-clicking on it and selecting Edit. The "Edit Advanced Area dialog opens, as shown in the following figure. You can add a name, description, select the type, and adjust the size and position.



Name:	SlowSectorTest			
Description:	This is a slow sector.			
Type:	Slow			
Angle:	0.0°			
Corner:	X:	12282	Y:	-8974
Opposite:	X:	16563	Y:	-4780

Sector Properties

There are other "sector" commands, such as: 'trackSectors', which provides the sectors at the robot's current position; 'trackSectorsAtGoal <goal>', which provides the sectors at a point; and 'trackSectorsAtPoint <X> <Y>', which provides the sectors for that point on the map. Note that all of these commands have identical output, which is shown in the Responses section.

For details on the related commands, see the Related Commands section.

Examples

The following example sends the robot to goal "g_10" and then lists the sectors that are in the path to that goal.

```
goto g_10
Going to g_10

tracksectorspath
```

```
Sectors over path
TrackSectors: SlowSector SlowSectorTest
End of TrackSectors

Arrived at g_10
```

The following example sends the robot to goal "g_10" and then lists the sectors that are within 500 mm in front of the robot on the path to that goal.

```
goto g_10
Going to g_10

tracksectorspath 500
Sectors over path for length 500
TrackSectors: SlowSector SlowSectorTest
End of TrackSectors

Arrived at g_10
```

Related Commands

trackSectors Command on page 279

trackSectorsAtGoal Command on page 282

trackSectorsAtPoint Command on page 285

trackSectorsPath Command on page 288

undock Command

Undocks the robot.

Syntax

undock

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
DockingState: <dock_state> ForcedState: <forced_state> ChargeState: <charge_state>
Stopping
Stopped
```

Details

The undock command tells the robot to move off of the dock/recharge station. It positions the robot in front of and facing the dock/recharge station.

When the robot is fully-charged, it will automatically undock from the dock/recharge station.

You can also undock the robot with one of the "goto..." commands. For details on these commands, use the links in the Related Commands section.

Examples

The following example undocks the robot:

```
undock
```

The command returns:

```
DockingState: Undocked ForcedState: Unforced ChargeState: Unknowable
Stopping
Stopped
```

Related Commands

goto Command on page 135

gotoPoint Command on page 137

gotoRouteGoal Command on page 139

dock Command on page 104

undock Command on page 291

updateInfo Command

Updates the value for and existing piece of information.

Syntax

updateInfo <infoName> <infoValue>

Usage Considerations

This ARCL command is only available on the robot.

You can only update information that was created with the createInfo command. For details, see createInfo Command on page 92.

Parameters

The command parameters are described in the following table.

Parameters	Definition
infoName	Enter the name for the information that you wish to update.
infoValue	Enter a string that represents the new information value.

Responses

The command returns:

```
Updated info for <infoName>
```

Details

This command is used to update the value of a piece of information that resides on the connected device. The information is initially created using the createInfo command. For details, see createInfo Command on page 92.

The updated information can be viewed using the getInfo command. For details, see getInfo Command on page 126 .

All information on the connected device can be listed with the getInfoList command. For details, see getInfoList Command on page 128.

Examples

To update the information called "myString" from an initial value of "testing" to a new value of "newtest", enter the following:

```
updateinfo myString newtest
```

The command returns:

```
Updated info for myString
```

Related Commands

createInfo Command on page 92

getInfo Command on page 126

getInfoList Command on page 128

waitTaskCancel Command

Cancels a wait task if one is active.

Syntax

waitTaskCancel

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
WaitState: <status>
```

The pauseTaskCancel command returns one of the following messages:

- WaitState: Waiting with status "Waiting"
- WaitState: Waiting interrupted
- WaitState: Waiting cancelled
- WaitState: Not waiting

These messages are broadcast to all of the clients, with the exception of "Not waiting".

Examples

The following example starts, builds and executes a task list that contains a "wait 10" task (3rd task) on the list. The waitTaskCancel command is used to cancel the wait task.

```
liststart mylist
List being cleared
Making new list

listadd goto g_5
Added task 'goto g_5' to the list
listadd goto g_6
Added task 'goto g_6' to the list
listadd wait 10
Added task 'wait 10' to the list
listadd goto g_23
Added task 'goto g_23' to the list

listexecute
Executing list

WaitState: Waiting 10 seconds with status "Waiting"
```

```
waittaskcancel
WaitState: Waiting completed

Successfully finished task list
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

waitTaskState Command

Displays the status of the wait task.

Syntax

waitTaskState

Usage Considerations

This ARCL command is only available on the robot.

Parameters

This command does not have any parameters.

Responses

The command returns:

```
WaitState: <status>
```

The waitTaskState command returns one of the following messages:

- WaitState: Waiting with status "Pausing"
- WaitState: Waiting interrupted
- WaitState: Waiting cancelled
- WaitState: Not waiting

These messages are **not** broadcast to all of the clients, with the exception of "Not waiting". This command is helpful for finding out what the current state of the robot is when connecting to ARCL.

Examples

The following example shows the status of the wait task.

```
waittaskstate
WaitState: Waiting with status "Waiting"
```

Related Commands

doTask Command on page 105

doTaskInstant Command on page 107

executeMacro Command on page 112

getMacros Command on page 130

listAdd Command on page 145

listExecute Command on page 147

listStart Command on page 149

pauseTaskCancel Command on page 183

pauseTaskState Command on page 185

waitTaskCancel Command on page 295

waitTaskState Command on page 297

ARCL Server Messages

The following table describes the server messages sent from ARCL to connected clients.

Server Message	Definition
Map changed	The map, with all of its related features, was just updated on the mobile robot.
Configuration changed	One or more ARAM configuration parameters was just updated on the mobile robot.
TextRequestChargeVoltage	This message is displayed when the battery voltage is below the LowBatteryVoltage threshold. When this occurs, the server message is displayed once per minute.
Estop pressed	The mobile robot motors were disabled.
Estop relieved	The mobile robot motors were enabled.
Motors disabled	The mobile robot motors were disabled, other than through an Estop. For example, using the LCD-interactive option.
Error: <error>	This message is displayed if an error occurs while a command is executing. For example: • Emergency stop pressed • Cannot find path • Failed going to goal • Stalled • Robot lost • Lost connection to robot • Server crashed
Interrupted: <command>	Commands may be interrupted. For example, if while going to goal2, you send the stop command, ARCL sends: "Interrupted: Going to goal2".
DockingState:	Includes docking and charge state information; this happens whenever there is a change to the docking state.

Robot Fault Messages

The following messages are broadcast to ARCL when robots set or clear faults.

Broadcast Message	Definition
Robot Fault	
Fault_Application	An application-specific fault has occurred. A description of the fault might be optionally provided by the application payload. The Enterprise Manager will not assign jobs to the robot when faults are present.
Driving_Application_Fault	An application-specific fault has occurred. A persistent popup will be displayed to the user. The robot will be unable to drive while this fault is asserted.
Critical OverTemperatureAnalog	The robot is too hot (measured by analog) and will shut down shortly.
Critical UnderVoltage	The robot battery is critically low and will shut down shortly.
EncoderDegraded	The robot's encoders may be degraded.
Critical GyroFault	The robot's gyro has had a critical fault. You may power-cycle the robot and continue using it, but you should also contact your robot provider for maintenance.
Robot Fault Cleared	
EncoderDegraded	The robot's encoders may be degraded.
Driving EncoderFailed	The robot's encoders have failed, turn off the robot and contact your robot provider for maintenance.
Critical GyroFault	The robot's gyro has had a critical fault, you may power cycle the robot and continue using it, but you should also contact your robot provider for maintenance.
Critical OverTemperatureAnalog	The robot is too hot (measured by analog) and will shut down shortly.
Critical UnderVoltage	The robot battery is critically low and will shut down shortly.
Critical_Application_Fault	An application-specific fault has occurred. A persistent popup will be displayed to the user.

See Also...

Introduction to ARCL on page 25
Enable Options in
Set ARCL Parameters in MobilePlanner on page 30
Connect to ARCL Using a Telnet Client on page 42
Using the ARCL Commands on page 46
ARCL Command Reference on page 70

See Also...

ARCL Server Messages on page 299