Aria Reference Manual 1.2.0

Generated by Doxygen 1.2.10

Fri Feb 28 18:56:36 2003

Contents

L	ARI	A overview	1
	1.1	Introduction	1
	1.2	ARIA, Java, Python, Saphira, Colbert and the ActivMedia Basic Suite	2
	1.3	License and Sharing	3
	1.4	The ARIA Package	3
	1.5	Documentation and Coding Convention	4
	1.6	ARIA Client-Server	4
	1.7	Robot Communication	5
	1.8	ArRobot	8
	1.9	Range Devices	11
	1.10	Commands and Actions	12
	1.11	Robot Callbacks	17
	1.12	Functors	17
	1.13	User Input	18
	1.14	ARIA Threading	19
	1.15	ARIA Global Data	20
	1.16	Piecemeal Use of ARIA	20
	1.17	Robot Parameter Files	21
	1.18	Utility Classes	22
	1.19	Sockets	23
	1.20	Non-everyday use of C++	24

ii CONTENTS

2	Aria Hierarchical Index					
	2.1 Aria Class Hierarchy	27				
3	Aria Compound Index	33				
	3.1 Aria Compound List	33				
4	Aria Class Documentation	39				
	4.1 ArAction Class Reference	39				
	4.2 ArActionAvoidFront Class Reference	43				
	4.3 ArActionAvoidSide Class Reference	45				
	4.4 ArActionBumpers Class Reference	47				
	4.5 ArActionConstantVelocity Class Reference	49				
	4.6 ArActionDesired Class Reference	51				
	4.7 ArActionDesiredChannel Class Reference	57				
	4.8 ArActionGoto Class Reference	58				
	4.9 ArActionGroup Class Reference	60				
	4.10 ArActionGroupInput Class Reference	63				
	4.11 ArActionGroupStop Class Reference	64				
	4.12 ArActionGroupTeleop Class Reference	65				
	4.13 Ar Action Group Unguarded Teleop Class Reference	66				
	4.14 ArActionGroupWander Class Reference	67				
	4.15 ArActionInput Class Reference	68				
	4.16 ArActionJoydrive Class Reference	70				
	4.17 ArActionKeydrive Class Reference	74				
	4.18 ArActionLimiterBackwards Class Reference	77				
	4.19 ArActionLimiterForwards Class Reference	79				
	4.20 ArActionLimiterTableSensor Class Reference	81				
	4.21 ArActionStallRecover Class Reference	83				
	4.22 ArActionStop Class Reference	85				
	4.23 ArActionTurn Class Reference	87				
	4.24 ArACTS_1_2 Class Reference	89				
	4.25 ArACTSBlob Class Reference	94				

CONTENTS

4.26 ArAMPTU Class Reference
4.27 ArAMPTUCommands Class Reference
4.28 Ar AMPTUPacket Class Reference
4.29 ArArg Class Reference
4.30 Ar Argument Builder Class Reference
4.31 ArArgumentParser Class Reference
4.32 Ar AsyncTask Class Reference
4.33 ArBasePacket Class Reference
4.34 Ar Commands Class Reference
4.35 Ar Condition Class Reference
4.36 ArDeviceConnection Class Reference
4.37 ArDPPTU Class Reference
4.38 ArDPPTUCommands Class Reference
4.39 ArDPPTUPacket Class Reference
4.40 ArFunctor Class Reference
4.41 ArFunctor1 Class Template Reference
4.42 ArFunctor1C Class Template Reference
4.43 ArFunctor2 Class Template Reference
4.44 ArFunctor2C Class Template Reference
4.45 ArFunctor3 Class Template Reference
4.46 ArFunctor3C Class Template Reference
4.47 ArFunctorASyncTask Class Reference
4.48 ArFunctorC Class Template Reference
4.49 ArGlobalFunctor Class Reference
4.50 Ar Global Functor 1 Class Template Reference
4.51 Ar Global Functor 2 Class Template Reference
4.52 Ar Global Functor 3 Class Template Reference
4.53 Ar GlobalRetFunctor Class Template Reference 182
4.54 ArGlobalRetFunctor1 Class Template Reference
4.55 ArGlobalRetFunctor2 Class Template Reference 187
4.56 ArGlobalRetFunctor3 Class Template Reference 191

4.57	ArGripper Class Reference
4.58	ArGripperCommands Class Reference
4.59	Aria Class Reference
4.60	ArInterpolation Class Reference
4.61	ArIrrfDevice Class Reference
4.62	ArJoyHandler Class Reference
4.63	ArKeyHandler Class Reference
4.64	ArLine Class Reference
4.65	ArLineSegment Class Reference
4.66	ArListPos Class Reference
4.67	ArLog Class Reference
4.68	ArLogFileConnection Class Reference
4.69	ArMath Class Reference
4.70	ArMode Class Reference
4.71	ArModeCamera Class Reference
4.72	ArModeGripper Class Reference
4.73	ArModeSonar Class Reference
4.74	ArModeTeleop Class Reference
4.75	ArModeUnguardedTeleop Class Reference
4.76	ArModeWander Class Reference
4.77	ArModule Class Reference
4.78	ArModuleLoader Class Reference
4.79	ArMutex Class Reference
4.80	ArNetServer Class Reference
4.81	ArNetServerConnection Class Reference
4.82	ArP2Arm Class Reference
4.83	ArPose Class Reference
4.84	ArPoseWithTime Class Reference
4.85	ArPref Class Reference
4.86	ArPriorityResolver Class Reference
4.87	ArPTZ Class Reference

CONTENTS

4.88 ArRangeBuffer Class Reference
4.89 ArRangeDevice Class Reference
4.90 Ar Range Device Threaded Class Reference $\ \ldots \ \ldots \ 320$
4.91 Ar Recurrent Task Class Reference
4.92 ArResolver Class Reference
4.93 ArRetFunctor Class Template Reference
4.94 ArRetFunctor1 Class Template Reference
4.95 ArRetFunctor1C Class Template Reference
4.96 ArRetFunctor2 Class Template Reference
4.97 ArRetFunctor2C Class Template Reference
4.98 ArRetFunctor3 Class Template Reference
4.99 ArRetFunctor3C Class Template Reference
$4.100 Ar Ret Functor C \ Class \ Template \ Reference \ \dots \ \dots \ 352$
4.101 Ar Robot Class Reference
$4.102 Ar Robot Packet \ Class \ Reference \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $
$4.103 Ar Robot Packet Receiver\ Class\ Reference\ \dots\ \dots\ 408$
$4.104 Ar Robot Packet Sender \ Class \ Reference \ \dots \ \dots \ \dots \ 411$
4.105ArRobotParams Class Reference
4.106ArSectors Class Reference
4.107 Ar Sensor Reading Class Reference
$4.108 Ar Serial Connection \ Class \ Reference \qquad . \ . \ . \ . \ . \ . \ . \ . \ . \ .$
4.109 Ar Sick Class Reference
4.110ArSickLogger Class Reference
4.111ArSickPacket Class Reference
4.112ArSickPacketReceiver Class Reference
4.113 Ar Signal Handler Class Reference
4.114ArSimpleConnector Class Reference
4.115ArSocket Class Reference
4.116ArSonarDevice Class Reference
4.117ArSonyPacket Class Reference
4.118ArSonyPTZ Class Reference

vi CONTENTS

4.119ArSyncTask Class Reference
4.120ArTaskState Class Reference
4.121ArTcpConnection Class Reference
4.122ArThread Class Reference
4.123ArTime Class Reference
4.124ArTransform Class Reference
4.125ArTypes Class Reference
4.126ArUtil Class Reference
4.127ArVCC4 Class Reference
4.128ArVCC4Commands Class Reference
4.129ArVCC4Packet Class Reference
4 130P2ArmJoint Class Reference 51

Chapter 1

ARIA overview

1.1 Introduction

Activ
Media Robotics Interface for Application (ARIA) Copyright 2002, Activ
Media Robotics, LLC. All rights reserved.

Welcome to ARIA. The software is an object-oriented, robot control applications-programming interface for ActivMedia Robotics' line of intelligent mobile robots.

Written in the C++ language, ARIA is client-side software for easy, high-performance access to and management of the robot server, as well as to the many accessory robot sensors and effectors. Its versatility and flexibility makes ARIA an excellent foundation for higher-level robotics applications, including SRI International's Saphira and the ActivMedia Robotics Basic Suite.

ARIA can be run multi- or single-threaded, using its own wrapper around Linux pthreads and WIN32 threads. Use ARIA in many different ways, from simple command-control of the robot server for direct-drive navigation, to development of higher-level intelligent actions (aka behaviors). For a description of how to integrate parts of ARIA with your other code, see **Piecemeal Use of ARIA** (p. 20).

This document contains an overview of ARIA. If you are browsing it in HTML, click a class or function link to view its detail pages. New users should view this document along with the ARIA examples.

You can download new versions of **Aria** (p. 205) from http://robots.activmedia.com/ARIA

1.2 ARIA, Java, Python, Saphira, Colbert and the ActivMedia Basic Suite

ARIA is for C++ object-oriented programmers who want to have close control of their robot. ARIA also is for those who have prepared robot-control software and want to quickly and easily deploy it on one or more ActivMedia Robotics mobile robot platforms.

ARIA now works in Java and Python! It has a Java wrapper and a Python wrapper included with the base release. This means that you can write ARIA programs in Java or Python as if ARIA itself was written in these languages. This wrapper is automatically generated by SWIG (http://www.swig.org) at each release, meaning that unlike three different implementations its consistent between languages, all three languages get new features and are maintained, and the many examples written for C++ ARIA are valid in the other languages.

There are a couple of more complicated/advanced features that don't work yet in these languages (you can use the classes in C++ that use the features, but you can't reimplement these features in the other languages). The only unimplemented feature of SWIG is virtual function overloading which means that you will not be able to make your own ArActions in Java or Python, but you can always add them to the C++ library and use them in Java or Python. For this deficiency the SWIG teams seems to be working on it so likely it will disappear in the future. You also will not be able to make your own ArFunctors for callbacks, but again where thats needed you can make objects in the C++ library and use them in Java or Python. For this deficiency language specific workarounds could likely be written by users, which I could incorporate or if there are large numbers of people using these wrappers we may develop these on our own. Also if the SWIG team solves virtual function overloading then simple classes to inherit from will remove the need for ArFunctors. Again though, you can use any of the existing modules in C++ that use these advanced features you just can't use these advanced features in Java or Python.

Look in the javaExamples/README.txt file for directions on how to use the Java wrapper and in the pythonExamples/README.txt for directions on how to use the python wrapper.

For creating applications with built-in advanced robotics capabilities, including gradient navigation and localization, as well as GUI controls with visual display of robot platform states and sensor readings, consider using SRI International's Saphira version 8 or later. Saphira v8 is built on top of ARIA, so you have access to all of ARIA's functionality, as well as its Saphira enhancements.

Non-programmers may create their own robot-control routines easily and simply with Saphira Colbert activity-building language. A Colbert editor, as well as some very advanced robot control applications including Navigator and World-Link, come in the Saphira/ARIA-based ActivMedia Basic Suite software. They

give you GUI access to all the features of your ActivMedia robot, including remote access across the global Internet.

Browse ActivMedia Robotics' support webpages http://www.activrobots.com and http://robots.activmedia.com for these and many other mobile robotics resources.

1.3 License and Sharing

ARIA is released under the GNU Public License, which means that if you distribute any work which uses ARIA, you must distribute the entire source code to that work. Read the included LICENSE text for details. We open-sourced ARIA under GPL not only for your convenience, but also so that you will share your enhancements to the software. If you wish your enhancements to make it into the ARIA baseline, you will need to assign the copyright on those changes to ActivMedia, contact aria-support@activmedia.com with these changes or with questions about this.

Accordingly, please do share your work, and please sign up for the exclusive ARIA-users@activmedia.com newslist so that you can benefit from others' work, too.

ARIA may be licensed for proprietary, closed-source applications. Contact sales@activmedia.com for details.

1.4 The ARIA Package

1.4.1 ARIA/

```
GPL license; agree to this to use ARIA
LICENSE
INSTALL
           Step-wise instructions for installing ARIA
README
           Also see READMEs in advanced/, examples/, and tests/
docs/
           Extensive documentation in HTML and PDF format
bin/
           Win32 binaries and dlls (Linux binaries in src/)
           ARIA examples -- a good place to start; see examples README
examples/
include/
           ARIA include files, of course
lib/
           Win32 .lib files and Linux .so files
           Robot definition (parameter) files (p2dx.p, for example)
params/
           ARIA source (*.cpp) files and Linux executables
src/
```

1.4.2 Other ARIA Files of Note

```
ARIA.dsp MSVC++ project file for building the ARIA libraries and examples ARIA.dsw Associated MSVC++ workspace for building ARIA and examples
```

Makefile Linux makefile for building ARIA and examples Makefile.dep Linux dependency

run Linux-only; builds and executes your ARIA application

tests/ Test files, somewhat esoteric but useful during ARIA development

utils/ Utility commands, not generally needed

advanced/ Advanced demos, not for the faint of heart (or ARIA novice)

1.5 Documentation and Coding Convention

For clarity while you read this technical document, we follow common C++ coding conventions:

1) Class names begin with a capital letter. 2) Enums either begin with a capital letter or are all in caps. 3) Avoid defines whenever possible. 4) Member variables in classes are prefixed with 'my'. 5) Static variables in classes are prefixed with 'our'. 6) Member function names start with a lower case. 7) Capitalize each word except the first one in a name; likeThisForExample. 8) Write all code so that it can be used threaded.

1.6 ARIA Client-Server

For those of you who are familiar with SRI International's Saphira software and ActivMedia Robotics' mobile robots and their related technologies, the underlying client-server control architecture for the mobile platform, sensors, and accessories hasn't changed much in ARIA. It's just gotten a lot better and more accessible.

The mobile servers, embodied in the Pioneer 2 and AmigoBot Operating System software and found embedded on the robot's microcontroller, manage the low-level tasks of robot control and operation, including motion, heading and odometry, as well as acquiring sensor information (sonar and compass, for example) and driving accessory components like the PTZ camera, TCM2 compass/inclinometer, and the Pioneer 5-DOF Arm. The robot servers do not, however, perform robotic tasks.

Rather, it is the job of an intelligent client running on a connected PC to perform the full gamut of robotics control strategies and tasks, such as obstacle detection and avoidance, sensor fusion, localization, features recognition, mapping, intelligent navigation, PTZ camera control, Arm motion, and much more. ARIA's role is on that intelligent client side.

Nearest the robot, ARIA's **ArDeviceConnection** (p. 124) class, at the behest of your application code, establishes and maintains a communication channel with the robot server, packaging commands to (**ArRobotPacketSender** (p. 411)) and decoding responses (**ArRobotPacketReceiver** (p. 408)) from the

robot in safe and reliable packet formats (ArRobotPacket (p. 406)) prescribed by the client-server protocols.

At its heart, ARIA's **ArRobot** (p. 355) class collects and organizes the robot's operating states, and provides clear and convenient interface for other ARIA components, as well as upper-level applications, to access that robot state-reflection information for assessment, planning, and ultimately, intelligent, purposeful control of the platform and its accessories.

ArRobot (p. 355)'s heart metaphor is particularly apt, too, since one of its important jobs is to maintain the clockwork cycles and multi-threaded rhythms of the robot-control system. Keyed to the robot's main information-packet cycle (hence, no longer a fixed timing cycle), ArRobot (p. 355)'s syncronous tasks (ArSyncTask (p. 475)) include the robot server-information packet handlers, sensor interpreters, action handlers, state reflectors, user tasks, and more. And your software may expand, replace, remove, and rearrange the list of synchronized tasks through ArRobot (p. 355)'s convenient sensor interp (Ar-Robot::addSensorInterpTask (p. 374)) and user task (ArRobot::addUser-Task (p. 375)) related methods.

Through its Action class, ARIA provides a flexible, programmable mechanism for behavior-level control of the robot server. An associated Resolver class lets you organize and combine actions, for coordinated motion control and intelligent guidance. With ARIA actions, you easily develop integrated guarded-teleoperation and color-blob tracking applications, for example.

ARIA also includes clear and convenient interface for applications to access and control ActivMedia Robotics accessory sensors and devices, including operation and state reflection for sonar and laser range finders, pan-tilt units, arms, inertial navigation devices, and many others.

The versatility and ease of access to ARIA code (sources included!) makes it the ideal platform for robotics client applications development.

1.7 Robot Communication

One of the most important functions of ARIA, and one of the first and necessary things that your application must do, is to establish and manage client-server communications between your ARIA-based software client and the robot's on-board servers and devices.

1.7.1 Connecting with a Robot or the Simulator

ArDeviceConnection (p. 124) is ARIA's communications object; **ArSerial-Connection** (p. 424) and **ArTcpConnection** (p. 481) are its built-in children most commonly used to manage communication between an ActivMedia robot

or the SRIsim robot simulator, respectively. These classes are not device-specific, however, so use **ArSerialConnection** (p. 424), for instance, to also configure a serial port and establish a connection with a robot accessory, such as with the SICK laser range finder.

You can also use a convenience class called **ArSimpleConnector** (p. 459) to do the connection for you, this is used in examples/demo, examples/wander, examples/teleop to name a few. This also will take and parse command line arguments so that you don't need to recompile to change where you want to connect. Among other benefits the **ArSimpleConnector** (p. 459) will try to connect to a simulator if one is running otherwise it'll connect to a serial port... so you don't have to recompile your program for either mode, just don't have a simulator running, or have one running.

Do note that some accessories, such as the P2 Gripper, PTZ camera, P2 Arm, compass, and others, which attach to the robot's microcontroller AUX serial port, are controlled through the client-side device connection with the robot. Use different methods and procedures other than **ArDeviceConnection** (p. 124) to communicate with and manage those devices through ARIA.

1.7.2 Opening the Connection

After creating and opening a device connection, associate it with its ARIA device handlers, most commonly with **ArRobot::setDeviceConnection** (p. 398) for the robot or the simulator.

For example, early in an ARIA program, specify the connection device and associate it with the robot:

```
ArTcpConnection (p. 481) con;
ArRobot (p. 355) robot;
```

Later in the program, after initializing the ARIA system (ARIA::Init(); is mandatory), set the Connection port to its default values (for TCP, host is "localhost" and port number is 8101), and then open the port:

```
con.setPort();
if (!con.openSimple())
  {
   printf("Open failed.");
   ARIA::shutdown();
   return 1;
}
```

TCP and Serial connections have their own implementation of open which is not inherited, but has default arguments that make the generic open work for the all default cases. And open returns a status integer which can be passed to the re-implemented and inherited **ArDeviceConnection::getOpenMessage** (p. 126) in order to retrieve related status string, which is useful in reporting errors to the user without having to know about the underlying device.

1.7.3 Robot Client-Server Connection

After associating the device with the robot, now connect with the robot's servers, **ArRobot::blockingConnect** (p. 377) or **ArRobot::asyncConnect** (p. 376), for example, to establish the client-server connection between ARIA **ArRobot** (p. 355) and the ActivMedia robot microcontroller or SRIsim simulated server. The blockingConnect method doesn't return from the call until a connection succeeds or fails:

```
robot.setDeviceConnection(&con);
if (!robot.blockingConnect())
{
   printf("Could not connect to robot... Exiting.");
   Aria::shutdown() (p.209);
   return 1;
}
```

The previous examples connect with the SRIsim simulator through a TCP socket on your PC. Use tcpConn.setPort(host, port) to set the TCP hostname or IP address and related socket number to another machine on the network. For instance, use tcpConn.setPort("bill", 8101); to connect to the Simulator which is running on the networked computer "bill" through port 8101.

Replace ArTcpConnection (p. 481) con; with ArSerialConnection (p. 424) con; to connect with a robot through the default serial port /dev/ttyS0 or COM1, or another you specify with con.setPort, such as con.setPort("COM3");.

At some point, you may want to open the port with the more verbose con.open();.

1.7.4 Connection Read, Write, Close and Timestamping

The two main functions of a device connection are **ArDevice-Connection::read** (p. 128) and **ArDeviceConnection::write** (p. 128). Simple enough. **ArDeviceConnection::close** (p. 126) also is inherited and important. You probably won't use direct read or write to the robot device, although you could. Rather, **ArRobot** (p. 355) provides a host of convenient methods that package your robot commands, and gather and distribute the various robot information packets, so that you don't have to attend those mundane details. See the next section for details.

All **ArDeviceConnection** (p. 124) subclasses have support for timestamping (**ArDeviceConnection::getTimeRead** (p. 127)). With the robot connection, timestamping merely says what time a robot SIP came in, which can be useful for interpolating the robot's location more precisely.

1.8 ArRobot

As mentioned earlier, **ArRobot** (p. 355) is the heart of ARIA, acting as client-server communications gateway, central database for collection and distribution of state-reflection information, and systems synchronization manager. **Ar-Robot** (p. 355) is also the gathering point for many other robot tasks, including syncTasks, callbacks, range-finding sensor and Actions classes.

1.8.1 Client Commands and Server Information Packets

Client-server communications between applications software and an ActivMedia robot or the Simulator must adhere to strict packet-based protocols. The gory details can be found in several other ActivMedia Robotics publications, including the Pioneer 2 Operations Manual and the AmigoBot Technical Manual. Suffice it to say here that **ArRobot** (p. 355) handles the low-level details of constructing and sending a client-command packets to the robot as well as receiving and decoding the various Server Information Packets from the robot.

1.8.2 Packet Handlers

Server Information Packets (SIPs) come from the robot over the robot-device connection and contain operating information about the robot and its accessories. Currently, there are two types of SIPs: the standard SIP and extended SIPs. The standard SIP gets sent by the robot to a connected client automatically every 100 (default) or 50 milliseconds. It contains the robot's current position, heading, translational and rotational speeds, freshly accumulated sonar readings, and much more. These data ultimately are stored and distributed by **ArRobot** (p. 355)'s State Reflection (see **State Reflection** (p. 10) below).

Extended SIPs use the same communication-packet protocols as the standard SIP, but with a different "type" specification and, of course, containing different operating information, such as I/O port readings or accessory device states like for the Gripper. And, whereas the standard SIP gets sent automatically once per cycle, your client controls extended packet communications by explicitly requesting that the server send one or more extended SIPs.

ArRobot (p. 355)'s standard SIP handler automatically runs as an **ArRobot** (p. 355) synchronized task. Other SIP handlers are built in, but your client must

1.8 ArRobot

add each to the connected robot object, and hence to the SIP handler sync task list, for it to take effect. See examples/gripperDemo.cpp for a good example.

You also may add your own SIP handler with **ArRobot::addPacketHandler** (p. 374). **ArListPos** (p. 230) keeps track of the order by which **ArRobot** (p. 355) calls each handler. When run, your packet handler must test the SIP type (**ArRobotPacket::getID** (p. 406)) and return true after decoding your own packet type or return false, leaving the packet untouched for other handlers.

1.8.3 Command Packets

From the client side going to the robot server, your ARIA program may send commands directly, or more commonly, use ARIA's convenience methods (Motion Commands and others) as well as engage Actions which ARIA ultimately converts into Direct Commands to the robot. See **Commands and Actions** (p. 12) for details. At the ARIA-robot interface, there is no difference between Action- or other ARIA convenience-generated commands and Direct Commands. However, upper-level processes aren't necessarily aware of extraneous Direct or Motion Commands your client may send to the robot. Motion Commands in particular need special attention when mixing with Actions. See **Commands and Actions** (p. 12) below for more details.

Once connected, your ARIA client may send commands to the robot server nearly at will, only limited by communication speeds and other temporal processes and delays. Similarly, the server responds nearly immediately with a requested SIP, such as a GRIPPERpac or IOpac which describe the P2 Gripper or Input/Output port states, respectively.

However, general information from the robot server about its odometry, current sonar readings, and the many other details which comprise its "standard" SIP automatically get sent to the ARIA client on a constant 100 or 50 millisecond cycle. This requires some synchronization with **ArRobot** (p. 355).

1.8.4 Robot-ARIA Synchronization

ArRobot (p. 355) runs a processing cycle: a series of synchronized tasks, including SIP handling, sensor interpretation, action handling and resolution, state reflection, and user tasks, in that order. By default, **ArRobot** (p. 355) performs these sequenced tasks each time it receives a standard SIP from the robot. Its cycle is thereby triggered by the robot so that the tasks get the freshest information from the robot upon which to act.

Of course, syncTasks runs without a connection with a robot, too. It has its own default cycle time of 100 milliseconds which you may examine and reset with **ArRobot::getCycleTime** (p. 385) and **ArRobot::setCycleTime** (p. 397), respectively. **ArRobot** (p. 355) waits up to twice that cycle time for

a standard SIP before cycling automatically.

ArRobot (p. 355)'s synchronization task list is actually a tree, with five major branches. If a particular task is not running, none of its children will be called. Each task has an associated state value and a pointer to an **ArTask-State::State** (p. 480) variable, which can be used to control the process, by turning it on or off, or to see if it succeeded or failed. If the pointer is NULL, then it is assumed that the task does not care about its state, and a local variable will be used in the task structure to keep track of that tasks state.

For each branch, tasks get executed in descending order of priority.

ARIA provides convenient methods to add your own sensor-interpretation and user tasks. Create an ARIA function pointer (Functors (p. 17)) and then add your sensor interpreter (ArRobot::addSensorInterpTask (p. 374)) or user task (ArRobot::addUserTask (p. 375)) to the list of syncTasks. These tasks can be removed; use ArRobot::remSensorInterpTask (p. 395) or Ar-Robot::remUserTask (p. 396) to remove sensor interpreter or user tasks, respectively, by name or by functor.

The intrepid ARIA programmer can add or prune branches from the **ArRobot** (p. 355) task list, as well as leaves on the branches. Do these things by getting the root of the tree with **ArRobot::getSyncTaskRoot** (p. 388), and then using the **ArSyncTask** (p. 475) class to do the desired manipulation.

You may disassociate **ArRobot** (p. 355)'s syncTask from firing when the standard SIP is received, through **ArRobot::setCycleChained** (p. 367). But in doing so, you may degrade robot performance, as the robot's cycle will simply be run once every **ArRobot::getCycleTime** (p. 385) milliseconds.

1.8.5 State Reflection

State reflection in the ArRobot (p. 355) class is the way ARIA maintains and distributes a snapshot of the robot's operating conditions and values, as extracted from the latest standard SIP. ArRobot (p. 355) methods for examining these values include ArRobot::getPose (p. 358), ArRobot::getX (p. 358), ArRobot::getY (p. 358), ArRobot::getTh (p. 358), ArRobot::getVel (p. 359), ArRobot::getRotVel (p. 359), Ar-Robot::getBatteryVoltage (p.359),ArRobot::isLeftMotorStalled (p. 359), ArRobot::isRightMotorStalled (p. 359), ArRobot::getCompass ArRobot::getAnalogPortSelected (p. 360), (p. 360). ArRobot::get-Analog (p. 360), ArRobot::getDigIn (p. 360), ArRobot::getDigOut (p. 360).

The standard SIP also contains low-level sonar readings, which are reflected in **ArRobot** (p. 355) and examined with the methods: **ArRobot::get-NumSonar** (p. 361), **ArRobot::getSonarRange** (p. 387), **ArRobot::getSonarNew** (p. 391), **ArRobot::getSonarReading** (p. 388), **ArRobot::get-**

ClosestSonarRange (p. 362), ArRobot::getClosestSonarNumber (p. 362). This information is more useful when applied to a range device; see Range Devices (p. 11) for details. And read the link pages for ArRobot (p. 355) state reflection method details.

ARIA's **ArRobot** (p. 355) also, by default, reflects in the State Reflection **Robot-ARIA Synchronization** (p. 9) syncTask the latest client Motion Command to the robot server at a rate set by **ArRobot::setState-ReflectionRefreshTime** (p. 401). If no command is in effect, the **Ar-Commands::PULSE** (p. 119) Direct Command gets sent. State reflection of the motion command ensures that the client-server communication watchdog on the robot won't time out and disable the robot.

You may turn the motion-control state reflector off in the **ArRobot::ArRobot** (p. 371) constructor (set doStateReflection parameter to false). This will cause Motion Commands to be sent directly to the robot whenever they are called. State Reflection will send a PULSE command to the robot at **ArRobot::get-StateReflectionRefreshTime** (p. 388) milliseconds to prevent the watchdog from timing out.

1.9 Range Devices

Range devices (**ArRangeDevice** (p. 312)) are abstractions of sensors for which there are histories of relevant readings. Currently, there are two ARIA RangeDevices: sonar (**ArSonarDevice** (p. 468)) and the SICK laser (**ArSick** (p. 431)). All range devices are range-finding devices that periodically collect 2-D data at specific global coordinates, so the RangeDevice class should work for any type of two-dimensional sensor.

Attach a RangeDevice to your robot with **ArRobot::addRangeDevice** (p. 363) and remove it with **ArRobot::remRangeDevice** (p. 395). Query for RangeDevices with **ArRobot::findRangeDevice** (p. 383). **ArRobot::has-RangeDevice** (p. 389) will check to see if a particular range device (the given instance) is attached to the robot. A list of range devices can be obtained with **ArRobot::getRangeDeviceList** (p. 387).

Note that sonar are integrated with the robot controller and that their readings automatically come included with the standard SIP and so are handled by the standard **ArRobot** (p. 355) packet handler. Nonetheless, you must explicitly add the sonar RangeDevice with your robot object to use the sonar readings for control tasks. ARIA's design gives the programmer ultimate control over their code, even though that means making you do nearly everything explicitly. Besides, not every program needs to track sonar data and there are some robots don't even have sonar.

Each RangeDevice has two sets of buffers (ArRangeBuffer (p. 305)): current and cumulative, and each support two different reading formats: box

and polar (ArRangeDevice::currentReadingPolar (p. 317), ArRangeDevice::currentReadingBox (p. 317), ArRangeDevice::cumulative-ReadingPolar (p. 316), ArRangeDevice::cumulativeReadingBox (p. 315)). The current buffer contains the most recent reading; the cumulative buffer contains several readings over time, limited by ArRangeBuffer::setSize (p. 311).

Useful for collision avoidance and other object detection tasks, apply the checkRangeDevices methods to conveniently scan a related buffer on all range devices attached to the robot for readings that fall within a specified range, including ArRobot::checkRangeDevicesCurrentPolar (p. 379), ArRobot::checkRangeDevicesCurrentBox (p. 379), ArRobot::checkRangesDevicesCumulativePolar, ArRobot::checkRangeDevicesCumulativeBox (p. 378).

Note that each range device also has a threading mutex (ArRange-Device::lockDevice (p. 318) and ArRangeDevice::unlockDevice (p. 319)) associated with it, so that sensors can be used in a thread-safe manner. For example, if a laser device gets added that runs in its own thread, the check-RangeDevice functions on the robot lock the device so it can poke at the laser device without running into any issues, unlocking the device when it is done. If you want to understand why this locking is good, see ARIA Threading (p. 19).

1.10 Commands and Actions

Your ARIA client drives the robot and runs its various accessories through Direct and Motion Commands, as well as through Actions.

1.10.1 Direct Commands

At the very lowest level, you may send commands directly to the robot server through **ArRobot** (p. 355). Direct commands consist of a 1-byte command number followed by none or more arguments, as defined by the robot's operating system, including P2OS and AmigOS. For example, the command number 4, aka ENABLE, enables the robot's motors if accompanied by the argument 1, and disables the motors with the argument 0.

Direct commands to the robot come in five flavors, each defined by its command argument type and length: Use **ArRobot::com** (p. 380) for commands that have no argument, such as PULSE; **ArRobot::comInt** (p. 381) for a 2-byte integer argument, signed or unsigned, such as the motors ENABLE command; **ArRobot::com2Bytes** (p. 380) for when you want to define each of the two bytes in the argument, such as the VEL2 command; and **ArRobot::comStr** (p. 381) or **ArRobot::comStrN** (p. 381) for a null-terminated or defined-length

(N extra argument) string argument, respectively, such as the sonar POLLING sequencing command.

The ArCommands (p. 119) class contains an enum with all the direct commands; ArCommands::ENABLE (p. 119), for example. Although identical in syntax and effect when supported, not all Direct Commands are included with every ActivMedia robot. Fortunately, unrecognized or otherwise malformed client commands are benign since they get ignored by the server. Please consult your robot's technical manual for details, such as the "Pioneer 2 Operating System" Chapter 6 in the Pioneer 2 Operations Manual, for client command numbers and syntax.

1.10.2 Motion Commands

At a level just above **ArRobot** (p. 355)'s Direct Commands are the Motion Commands. These are explicit movement commands. Some have identical Direct Command analogues and act to immediately control the mobility of your robot, either to set individual-wheel, or coordinated translational and rotational velocities (**ArRobot::setVel2** (p. 402), **ArRobot::setVel** (p. 401), **ArRobot::setRotVel** (p. 401), respectively); change the robot's absolute or relative heading (**ArRobot::setHeading** (p. 400) or **ArRobot::setDelta-Heading** (p. 398), respectively); move a prescribed distance (**ArRobot::move** (p. 392)); or just stop (**ArRobot::stop** (p. 402)).

Examine the directMotionDemo.cpp example file to to see Motion Commands at work.

Be aware that a Direct or a Motion Command may conflict with controls from Actions or other upper-level processes and lead to unexpected consequences. Use **ArRobot::clearDirectMotion** (p. 380) to cancel the overriding effect of a Motion Command so that your Action is able to regain control the robot. Or limit the time a Motion Command prevents other motion actions with **Ar-Robot::setDirectMotionPrecedenceTime** (p. 399). Otherwise, the Motion Command will prevent actions forever. Use **ArRobot::getDirectMotion-PrecedenceTime** (p. 386) to see how long a Motion Command takes precedence.

1.10.3 Actions

Instead of using Direct or Motion Commands, we prefer that your ARIA client software use Actions to drive the robot. **ArAction** (p. 39) is the base class; **ArAction::fire** (p. 42) is the only function that needs to be overloaded for an action to work. ARIA includes a number of built-in actions; look for them in the ARIA sources (the inheritance diagram on the **ArAction** (p. 39) page will show you which they are as well). And see the actionExample program to discover how to create your own actions.

Actions are added to robots with **ArRobot::addAction** (p. 371), including a priority which determines its position in the action list. **ArAction::setRobot** (p. 41) is called on an action when it is added to a robot. You can override this. For example, this would be useful to add a connection callback, if there were some calculations you wished to do upon connection to the robot.

Actions are evaluated by the resolverin descending order of priority (lowest priority goes last) in each **ArRobot** (p. 355) syncTask cycle just prior to State Reflection. The resolver goes through the actions to find a single end action-Desired (**ArActionDesired** (p. 51)). Depending on its current state, an action contributes particular actionDesired movement values and strengths to the final action desired. After this final action desired has been calculated, it is stored and later gets passed to the State Reflector and on to the robot as motion commands.

At each stage when the resolver is evaluating an action it passes in the current action desired of the higher priority actions, this is the currentDesired. For example, a stall-recovery action probably should be programmed not to exert its motion effects if it has been pre-empted by a stop action, so the stall-recovery action can check and see if either the strength is used up or if there is a maximum velocity, and if so it can reset its state. However, there is no need for an action to pay attention to the currentDesired. The resolver could also simply pass a ArActionDesired.reset() to the actions if it did not want the actions to know about its state.

1.10.4 Action Desired

ArActionDesired (p. 51) is the meat of actions. Desired actions should be reset (**ArActionDesired::reset** (p. 51)) before they are used or reused.

There are six desired action channels: velocity (ArActionDesired::setVel (p. 56)), relative heading (ArActionDesired::setDeltaHeading (p. 55)), absolute heading (ArActionDesired::setHeading (p. 55)), maximum forward translational velocity (ArActionDesired::setMaxVel (p. 56)), maximum reverse translational velocity (ArActionDesired::setMaxNegVel (p. 55)), and maximum rotational velocity (ArActionDesired::setMaxRotVel (p. 55)).

Your action gives each channel a strength of 0.0, the lowest, to 1.0, the highest. Strengths are used by the resolver to compute the relative effect the action-Desired channel setting will have on the current translational velocity and heading of the robot, as well as the speed limits for those movements. (Note that deltaHeading and heading are treated as the same channel for strength purposes, and that these are simply alternate ways of accessing the same channel.)

The maximum velocity, maximum negative velocity, and maximum rotational velocity channels simply impose speed limits and thereby indirectly control the robot.

For more advanced usage, desired actions can be merged (ArAction-Desired::merge (p. 54)) and averaged (ArActionDesired::startAverage (p. 56), ArActionDesired::addAverage (p. 54), ArActionDesired::end-Average (p. 54)).

1.10.5 Resolvers

ArResolver (p. 325) is the base action-resolver class. **ArPriorityResolver** (p. 297) is the default resolver. **ArResolver::resolve** (p. 325) is the function that **ArRobot** (p. 355) calls with the action list (actually **Ar-Resolver::ActionMap** (p. 325)) in order to combine and thereby resolve the actionDesired movement controls into State Reflection motion commands to the robot server.

There may only be one resolver per robot, which is set with **ArRobot::set-Resolver** (p. 366). However, a resolver could be created to contain multiple resolvers of its own. Also note that though a robot has particular resolver bound to it, a resolver instance is not tied to any robot. Thus, if you had some adapative resolver, you could set it to work for all robots.

The resolver works by setting each of the currentDesired channels to the contributing actionDesired values in proportion to their respective strengths and priority, adjusting each movement channel's currentDesired value until the individual strength becomes 1.0 or the list is exhausted. Same-priority actions get averaged together (if they are competing) before being resolved with higher-priority results.

The following table illustrates the steps and currentDesired setVel results when the resolver combines four fictional actionDesired setVel channel values and their relative strengths:

step #	action	priority	Desired::setVel	strength	currentDesired	strength
1	4	4	-400	0.25	-400	0.25
2	3	3	-100	1.0	(combine to 2&3)	
3	2	3	200	0.50	(combine to 2&3)	
4	2&3	3	0	0.75	-100	1.0
5	1	1	500	0.50	-100	1.0

Notice in the example that the same-priority actions 2 and 3 are combined before being resolved with the higher priority action 4. Also notice that action 1 has no effect since the currentDesired channel strength reaches 1.0 before that lowest-priority action gets considered by the resolver.

1.10.6 Movement and Limiting Actions

For programming convenience, ARIA has defined two useful types of actions: Movement and Limiting. There are no classes for limiting or movement actions.

Built in movement actions have an **ArAction** (p. 39) prefix and act to set either or both the translational velocity (setVel) and heading (setDeltaHeading and setHeading) channels. Built in limiting actions are prefixed with ArAction-Limiter and act to set one or more of the maximum translational and rotational velocity channels.

1.10.7 Mixing Actions

Actions are most useful when mixed. The teleop program is a good example of mixing limiting and movement actions. In the code, there are many limiting actions, including Limiter, LimiterFar, and so on. And there are two movement actions, joydriveAct and keydriveAct. The limiting actions have higher priority than the movement ones, thereby making sure the way is safe before allowing the robot to drive.

The example also illustrates fundamental, yet very powerful features of ARIA actions and how they contribute to the overall behavior of the mobile robot. Because they are individuals, contributing discretely to the movements of the robot, actions are easily reusable. The limiting action in the teleop example that prevents the robot from crashing into a wall when translating forward, can be copied, as is, into another ARIA program and have the identical effect, except that instead of driving the robot with a joystick, the new program's lower-priority movement action might use color-tracking to have the robot follow a rolling ball. The ball-following action doesn't needs to know anything about the finer arts of safe navigation—the higher-priority limiting actions take care of that

Another ARIA example program called wander.cpp demonstrates how different movement actions can be used and how they interact. The stall-recover action in wander (ArActionStallRecover (p. 83)) influences the robot's movements only when the motors are stalled, disabling the lower priority actions by using up all translational and rotational strength until the robot has extracted from the stall. You should also examine ArActionStallRecover.cpp in the src/ directory for actionDesired details on how the action changes its motion control influences based on the stall state.

Also note how **ArActionAvoidFront** (p. 43) and **ArActionConstant-Velocity** (p. 49) interact.

1.11 Robot Callbacks 17

1.11 Robot Callbacks

There are a number of useful callbacks in the ARIA system, including Ar-Robot::addConnectCB (p. 372), ArRobot::remConnectCB (p. 393), Ar-Robot::addFailedConnectCB (p. 373), ArRobot::remFailedConnectCB (p. 394), ArRobot::addDisconnectNormallyCB (p. 372), ArRobot::remDisconnectNormallyCB (p. 394), ArRobot::addDisconnectOnErrorCB (p. 373), ArRobot::remDisconnectOnErrorCB (p. 394), ArRobot::add-RunExitCB (p. 374), ArRobot::remRunExitCB (p. 395). Read their individual documentation pages for details.

Examples of callbacks are in the directMotionDemo and in joydriveThreaded. Also, **ArGripper** (p. 196) uses a connectCB as a way to find out when to poll the robot – a good use of callbacks. Just make sure that any modular code you have removes callbacks if you use them.

1.12 Functors

Functor is short for function pointer. A Functor lets you call a function without knowing the declaration of the function. Instead, the compiler and linker figure out how to properly call the function.

Function pointers are fully supported by the C language. C++ treats function pointers like C, but adds in the concept of member functions and the 'this' pointer. C++ does not include the 'this' pointer in the function pointer, which can cause all sorts of problems in an object-oriented program. Hence, we created functors. Functors contain both the function pointer and the pointer to the object which contains the function, or what the function uses as its 'this' pointer.

ARIA makes use of functors as callback functions. In most cases, you will only need to instantiate callback functors and pass them off to various parts of ARIA. To instantiate a functor, you first need to identify how many parameters the function needs and if it returns a value. Most ARIA functions take a pointer to **ArFunctor** (p. 139). This is the base class for all the different functors. Its for a function that has no parameters and no return value.

But you can not create an **ArFunctor** (p. 139), because it is an abstract base class. Rather, you need to instantiate one of these classes:

ArFunctorC (p. 165), ArFunctor1C (p. 144), ArFunctor2C (p. 150), ArRetFunctorC (p. 352), ArRetFunctor1C (p. 330), ArRetFunctor2C (p. 336)

The 'C' in the name means that it's an instance of the functor that knows about the class of a member function. These are templatized classes so need to be instantiated. For example:

```
ExampleClass obj;
ArFunctorC (p. 165) < ExampleClass > functor(obj, & ExampleClass:: a Function);
```

ExampleClass is a class which contains a function called aFunction. Once the functor is created in this fashion, it can now be passed off to an ARIA function that wants a callback functor. And the function ExampleClass::aFunction will be called when the functor is invoked.

The code that uses the callback functor only needs to know about these templatized classes: **ArFunctor** (p. 139), **ArFunctor1** (p. 142), **ArFunctor2** (p. 148), **ArRetFunctor** (p. 327)<ReturnType>, **ArRetFunctor1** (p. 328)<ReturnType>, and **ArRetFunctor2** (p. 334)<ReturnType>. These functors take 0-2 parameters and have no return or a return value.

To invoke the functors, simply call the invoke function on the functor. If it takes parameters, call invoke with those parameters. If the functor has a return value, call invokeR. The return value of the function will be passed back through the invokeR function.

1.13 User Input

There are two different ways to get user input into Aria (p. 205), from a joystick and from a keyboard. With a joystick is most useful for driving the robot around. There is a class set up that interfaces to the OS for joystick controls, this is Ar-JoyHandler (p. 214), the important functions are ArJoyHandler::getButtons, ArJoyHandler::getAdjusted (p. 216), ArJoyHandler::setSpeeds (p. 214), and ArJoyHandler::getDoubles (p. 217). With a keyboard is most useful for setting and changing modes, and exiting the program, but it it can also be used to drive the robot as well (with the arrow keys and the space bar typically), Ar-**KeyHandler** (p. 220) is the class which deals with interfacing to the keyboard. ArKeyhandler is directed towards capturing single key presses, not towards reading in sets of text, you can use the normal OS functions to do this. The important functions in ArKeyHandler (p. 220) is ArKeyHandler::addKey-Handler (p. 222), which binds a specific key to a given functor, also look at the enum ArKeyHandler::KEY (p. 221) for values to pass in for special keys. You also need to attach a key handler to some robot with ArRobot::attachKey-Handler (p. 377). ArActionJoydrive (p. 70) will use the joystick to drive the robot around, while **ArActionKeydrive** (p. 74) will use the arrow keys and spacebar to drive the robot around, both of these are employed in the teleop example. The keyboard control is also a nice way to exit cleanly in Windows since control C or just clicking on the program box won't cleanly disconnect from the robot, by default if you connect an ArKeyHandler (p. 220) to a robot, escape will exit the program, however you can chage this behavior when you attach the key handler to the robot if you wish.

1.14 ARIA Threading

ARIA is highly multi-threaded. This section presents some of the critical concepts behind writing threaded ARIA code.

ARIA provides a number of support classes to make it easier to write object-oriented threaded code. They are: **ArASyncTask** (p. 110), **ArCondition** (p. 122), **ArMutex** (p. 266), and **ArThread** (p. 487).

Thread-safe code mostly means proper coordination between threads when handling the same data. You want to avoid the obvious problem of one or more threads reading the data at the same time others write the data. To prevent this problem from happening, the data needs to be protected with synchronization objects.

1.14.1 Synchronous Objects

In ARIA, the synchronization objects are **ArMutex** (p. 266) and **ArCondition** (p. 122). **ArMutex** (p. 266) is the most useful one. Mutex is short for mutual exclusion. It guarantees that only one thread will use its data at a time. The proper way to use a mutex is to attempt to lock it right before accessing its shared data. If the mutex is not in use, ARIA then grants exclusive access by the requesting thread. If the mutex is locked, the access request gets blocked, and the thread must wait until the mutex gets free. When the thread that has access to the mutex is finished with the data, it must unlock the mutex and thereby make the data available to other threads. If it is not unlocked, the program will become deadlocked and hang. See the mutex example in the ARIA distribution for more details.

ArCondition (p. 122) is useful for delaying the execution of a thread. A thread suspends execution while waiting on an **ArCondition** (p. 122) until another thread wakes it up. For instance, use **ArCondition** (p. 122) while waiting for a mutex to become free. Performance is better, too. **ArCondition** (p. 122) puts the thread to sleep. The processing expensive alternative is to have the thread continously check for a change in condition. **ArCondition** (p. 122) notifies only those threads that are currently waiting on it at the time the condition changes.

See the ARIA condition example.

1.14.2 Asynchronous Tasks

Unlike the cyclical tasks in the syncTask list, asynchronous tasks run in their own threads. And an ARIA **ArASyncTask** (p. 110) needs to have a thread under its control for the full lifetime of the program.

To create an ansynchronous task, derive a class from ArASyncTask (p. 110)

and override the ArASyncTask::runThread() (p. 111) function. (The function automatically is called within the new thread, when the ArASyncTask (p. 110) gets created.) To create and start the thread, call ArASyncTask::create() (p. 110). When the ArASyncTask::runThread() (p. 111) function exits, the thread will exit and be destroyed.

This class is mainly a convenience wrapper around **ArThread** (p. 487) so that you can easily create your own object that encapsulates the concept of a thread.

1.15 ARIA Global Data

ARIA contains a list of all the **ArRobot** (p. 355) instances. Use the ARIA::find-Robot() to find a robot by name, or use ARIA::getRobotList() to get a list of the robots.

Use ARIA::getDirectory() to find ARIA's top-level path (C:\Aria (p. 205) or /usr/local/Aria (p. 205), typically). This is useful, for instance, to locate robot parameter files for individual operational details. Use ARIA::setDirectory() to change this path for the run of the program if you feel the need to override what Aria (p. 205) has decided.

1.16 Piecemeal Use of ARIA

The most basic layer of ARIA is its deviceConnections, which handles low-level communication with the robot server. On top of the connection layer, we have a packet layer—**ArBasePacket** (p. 112) and **ArRobotPacket** (p. 406)—the basic algorithms for constructing command packets and decoding server information packets.

Above the packet layer is the packet handler classes, **ArRobotPacketReceiver** (p. 408) and **ArRobotPacketSender** (p. 411), when send and receive packets to and from the robot. Finally, on top of all these lowest layers is **ArRobot** (p. 355), which is a gathering point for all things, but can be used in a quite basic format without all of the bells and whistles. **ArRobot** (p. 355) has builtin tasks, actions, state reflection and so forth, all of which can be disabled from the constructor (**ArRobot::ArRobot** (p. 371)) and ignored or reimplemented.

Also note that if all you do is turn off state reflection, which only affects sending **ArRobot** (p. 355)-mediated motion commands to the robot, not receiving SIPs from the robot, none of the other activities which **ArRobot** (p. 355) engages on its loop will take up hardly any time, so it probably isn't worth building your own set of tasks, but the power to do so is there for the intrepid.

One other thing worth noting is that you can call **ArRobot::loopOnce** (p. 392) and it will run through its loop a single time and return. This is so that you can

use ARIA from your own control structure. If you are using loopOnce you may also find it beneficial to call **ArRobot::incCounter** (p. 368), so that the loop counter will be updated. You could also just call **ArRobot::packetHandler** (p. 393), **ArRobot::actionHandler** (p. 371), or **ArRobot::stateReflector** (p. 402) on your own, as these are the most important internal functions, though if you make your own loop you should probably call **ArRobot::incCounter** (p. 368) any way that you do it, as this is how sonar are known to be new or not, and such.

We recommend that whatever you do you use the same type of strict threading/locking that ARIA observes.

1.17 Robot Parameter Files

Found in the Aria (p. 205)/params directory, generic, as well as individually named robot parameter files contain default and name-specific robot information that ARIA uses to characterize the robot and correctly interpret the server information that a robot sends back to the client.

Every robot has a type and subtype, such as Pioneer and P2AT, as well as a user-modifiable name, embedded in its FLASH parameters. These parameters get sent to the ARIA client right after establishment of the client-server connection. ARIA retrieves parameter files in the following order—built in defaults, subtype parameter file, and finally name parameter file—setting and resetting global variables based on the contents of each file. Accordingly, subtype may add or change the settings derived from the default, and a named parameter file has the very last say over things.

ARIA has default generic type parameters, and generic subtype robot files, such as p2at.p, p2de.p or p2pp.p for the Pioneer 2-AT, and Pioneer 2-DE and Performance PeopleBot subtypes, respectively, in the parameters directory. You may change their contents to better match your specific robot. Or, better, either create a new one or copy the contents to a file which name matches your robot's FLASH parameter name, adding the ".p" parameter file suffix. Then change and add to the generic factors section those accessory or other operational details that best define that specific robot.

For example, ARIA uses RobotRadius to determine the robot's turn limits in most of the obstacle avoidance routines. The default for the P2AT robot doesn't account for bumper accessories. Accordingly, you might create a new parameter file that redefines RobotRadius for that specific robot.

ARIA uses the values in the conversion factors section of a parameter file to transform the robot-dependent server information data into normal dimensions and rates. For example, the DistConvFactor converts the robot's position data, measured in encoder ticks, into millimeters.

ARIA consults the accessories section of a robot's parameter file to determine what accessories a robot might have that cannot be told by other means. For example, the P2 bumper values appear in the standard SIP stall values, but if a bump ring isn't connected, these values float and vacillate between on and off. An accessory definition in the parameter file clues ARIA to use or not use the bumper values.

Finally, the sonar section of the parameter file contains information about the sonar number and geometry so that ARIA can relate sonar readings with position relative to the center of the robot.

1.17.1 How the parameter file works

The parameter file is very much like a Windows INI file in format. It contains sections and keyword/data pairs. Comments start with a semi-colon. A section identifier is a bracketed keyword, such as:

```
[ConvFactors]
```

Keywords and data are separated by one or more spaces on a single line, and may include several defining data values. Each keyword has its own behavior with how it parses the data. For example:

```
KeyWord data1 data2 data3 ...
```

Case matters for both section identifiers and keyword names. Some parameters can have multiple instances in the file. SonarUnit is a good example of this. The multiple instances of the parameter need to be surrounded by a '@start' and '@end' block. For example:

```
@start
SonarUnit 0 73 105 90
SonarUnit 1 130 78 41
@end
```

See ArPreferences.h for additional details.

1.18 Utility Classes

Some of the utility classes are **ArMath** (p. 238), **ArUtil** (p. 497), **ArTime** (p. 491), **ArPose** (p. 285), and **ArSectors** (p. 418).

1.19 Sockets 23

1.19 Sockets

The **ArSocket** (p. 461) class is a wrapper around the socket network communication layer of your operating system. ARIA mostly uses **ArSocket** (p. 461) to open a server port and to connect to another server port.

To connect to a port, simply construct a socket containing the hostname or IP address of the host, a port number, and the ARIA socket type (TCP or UDP). For example:

```
ArSocket (p.461) sock("host.name.com", 4040, ArSocket::TCP);
Or call the ArSocket::connect() (p.461) function, such as:
ArSocket (p.461) sock;
    sock.connect("host.name.com", 4040, ArSocket::TCP);
To open a server port, simple construct a socket:
    ArSocket (p.461) sock(4040, true, ArSocket::TCP);
Or call:
    ArSocket::open(4040, ArSocket::TCP);
```

1.19.1 Emacs

Here is the configuration specification the developers at ActivMedia Robotics use in their .emacs files, in case you want to modify the code using emacs and not deal with differences in indentation and such.

```
(setq c-default-style '((other . "user")))
(c-set-offset 'substatement-open 0)
(c-set-offset 'defun-block-intro 2)
(c-set-offset 'statement-block-intro 2)
(c-set-offset 'substatement 2)
(c-set-offset 'topmost-intro -2)
(c-set-offset 'arglist-intro '++)
(c-set-offset 'statement-case-intro '*)
(c-set-offset 'member-init-intro 2)
(c-set-offset 'inline-open 0)
(c-set-offset 'brace-list-intro 2)
(c-set-offset 'statement-cont 0)
(defvar c-mode-hook 'c++-mode)
```

1.20 Non-everyday use of C++

1.20.1 Standard Template Library

ARIA makes heavy use of the C++ standard template library. So you should understand the STL in order to get the best use from some of the more advanced parts of ARIA. A reference many developers have found useful is http://www.sgi.com/tech/stl/, this is documentation to SGI's implementation, but other than the SGI specific templates which are explicitly stated as being SGI only, the documentation is quite helpful.

1.20.2 Default Arguments

Default arguments work like the following, in the function delcaration a parameter is specified, and given a default value at the same time. If the function is then used the parameters which have been given a value do not need to be given values when the function is used.

For example, after defining foo, it can be used in two differnt manners:

```
void foo(int number = 3);
// ...later
foo();
// or
foo(int);
```

This behavior is quite useful for having defaults that most people will not need to change, but allowing people to change them if they desire.

Also note that the function definition must not have the assignment in it, only the declaration, otherwise Windows compilers will not work and will report a not entirely useful error message.

1.20.3 Constructor Chaining

Constructor chaining is quite simple though little used. Each contructor can give arguments to the constructors of the member variables it contains and to the constructors which it inherits. For example if you have:

```
class BaseClass
{
  public:
    BaseClass(int someNumber);
};
and
```

```
class SubClass : public BaseClass
{
public:
   SubClass(void);
   int anotherNumber;
};
```

When you write your constructor for subClass. you can intialize both baseClass and anotherNumber:

```
SubClass::SubClass(void) : BaseClass(3), anotherNumber(37)
{
    // ...
}
```

Note how the constructors to be initialized must follow a colon (:) after the constructor, and be separated by commas? For member variables they must also be initialized in the order they are in the class. Note that initializing integers is not all that unique or useful, but using this to initialize callback **Functors** (p. 17) is quite useful.

Constructor chaining is used in many many places by ARIA, thus it must be understood in order to understand ARIA, but the above is all that really needs to be known.

1.20.4 Chars and Strings, Win workaround

During development problems were encountered with windows if std::strings were passed into a dll. Thus for all input to ARIA const char *s are used, but for all internal storage and all reporting std::strings are passed back out of ARIA.

1.20.5 AREXPORT

Because of the Windows set up for using DLLs, is a macro used to take care of the requirements for DLLs. Largely users do not need to worry about AREXPORTs, but only functions which have AREXPORTs or inline functions are usable with DLLs in windows (all of the functions which are documented are usable).

Chapter 2

Aria Hierarchical Index

2.1 Aria Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:
ArAction
ArActionAvoidFront
ArActionAvoidSide
ArActionBumpers
ArActionConstantVelocity
ArActionGoto
ArActionInput
ArActionJoydrive
ArActionKeydrive
ArActionLimiterBackwards
ArActionLimiterForwards
ArActionLimiterTableSensor
ArActionStallRecover
ArActionStop
ArActionTurn
ArActionDesired
ArActionDesiredChannel
ArActionGroup
ArActionGroupInput
ArActionGroupStop
ArActionGroupTeleop
ArActionGroupUnguardedTeleop
ArActionGroupWander
ArACTS_1_2
ArACTSBlob

ArAMPTUCommands	99
ArArg	103
ArArgumentBuilder	106
ArArgumentParser	107
ArBasePacket	112
ArAMPTUPacket	101
ArDPPTUPacket	138
ArRobotPacket	406
ArSickPacket	447
ArSonyPacket	470
ArVCC4Packet	514
ArCommands	119
ArCondition	122
ArDeviceConnection	124
ArLogFileConnection	233
ArSerialConnection	424
ArTcpConnection	481
$\label{eq:ardpptu} \mbox{ArDPPTUCommands} \ \dots $	136
ArFunctor	139
$ArFunctor1 < P1 > \dots $	142
$ArFunctor1C < T, P1 > \dots \dots \dots \dots \dots \dots$	144
$ArGlobalFunctor1 < P1 > \dots \dots \dots \dots \dots$	170
$ArFunctor 2 < P1, P2 > \dots \dots \dots \dots \dots \dots \dots \dots$	148
$ArFunctor2C < T, P1, P2 > \dots \dots \dots \dots$	150
$ArGlobalFunctor2 < P1, P2 > \dots \dots \dots \dots \dots$	173
ArFunctor $3 < P1, P2, P3 > \dots$	155
ArFunctor3C $<$ T, P1, P2, P3 $>$	157
ArGlobalFunctor $3 < P1, P2, P3 > \dots \dots \dots \dots$	177
$ArFunctorC < T > \dots \dots \dots \dots \dots \dots \dots \dots \dots$	165
ArGlobalFunctor	168
$ArRetFunctor < Ret > \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots$	327
$ArGlobalRetFunctor < Ret > \dots \dots \dots \dots \dots$	182
$ArRetFunctor1 < Ret, P1 > \dots \dots \dots \dots \dots$	328
$ArGlobalRetFunctor1 < Ret, P1 > \dots \dots \dots \dots$	
$ArRetFunctor1C < Ret, T, P1 > \dots \dots \dots \dots$	
$ArRetFunctor2 < Ret, P1, P2 > \dots \dots \dots \dots \dots$	334
$ArGlobalRetFunctor2 < Ret, P1, P2 > \dots \dots \dots$. 187
$ArRetFunctor2C < Ret, T, P1, P2 > \dots \dots \dots \dots$. 336
$ArRetFunctor3 < Ret, P1, P2, P3 > \dots \dots \dots \dots$	342
$ArGlobalRetFunctor3 < Ret, P1, P2, P3 > \dots \dots$. 191
$ArRetFunctor3C < Ret, T, P1, P2, P3 > \dots \dots \dots \dots$. 345
$ArRetFunctorC < Ret, T > \dots \dots \dots \dots \dots$	352
ArGripper	196
ArGripperCommands	203

Aria ArInterpolation ArJoyHandler ArKeyHandler ArLine ArLineSegment ArListPos ArLog ArMath ArMode ArModeBumps ArModeCamera	205 210 214 220 224 226 230 231 238 244
ArModeGripper	250
ArModeIO	
ArModeLaser	
ArModePosition	
ArModeSonar	252
ArModeTeleop	254 256
ArModeWander	258
ArModule	260
ArModuleLoader	263
ArMutex	266
ArNetServer	268
ArNetServerConnection	271
ArP2Arm	273
ArPose	285
ArPoseWithTime	289
ArPref	290
ArPreferences	
ArRobotParamFile ArRobotAmigo	
ArRobotGeneric	
ArRobotMapper	
ArRobotP2AT	
ArRobotP2AT8	
ArRobotP2AT8Plus	
ArRobotP2CE	
ArRobotP2D8	
ArRobotP2D8Plus	
ArRobotP2DF ArRobotP2DX	
ArRobotP2DXe	
ArRobot 2DXC	
ArRobotP2PB	

ArRobotP2PP
ArRobotPerfPB
ArRobotPerfPBPlus
ArRobotPion1M
ArRobotPion1X
$\operatorname{ArRobotPionAT}$
ArRobotPowerBot
ArRobotPsos1M
ArRobotPsos1X
ArRobotPsos43M
ArPrefSection
ArPTZ
ArAMPTU
ArDPPTU
ArSonyPTZ
ArVCC4
ArRangeBuffer
ArRangeDevice
ArIrrfDevice
ArRangeDeviceThreaded
ArSick
ArSonarDevice
ArResolver
ArPriorityResolver
ArRobot
ArRobotPacketReceiver
ArRobot Packet Sender
ArRobot Params
ArSectors
ArSensorReading
ArSickLogger
ArSickPacketReceiver
ArSimpleConnector
ArSocket
ArSyncTask
ArTaskState
ArThread
ArASyncTask
v
ArFunctorASyncTask
ArRecurrentTask
ArSignalHandler
ArSyncLoop
ArTime
ArTransform

ArTypes															496
$\operatorname{ArUtil} \ \dots \ \dots \ \dots$															497
ArVCC4Commands															512
P2ArmJoint															515

Chapter 3

Aria Compound Index

3.1 Aria Compound List

lere are the classes, structs, unions and interfaces with brief descriptions:	
$\mathbf{ArAction}$ (Action class, what typically makes the robot move)	39
ArActionAvoidFront (This action does obstacle avoidance, control-	
ling both trans and rot)	43
ArActionAvoidSide (Action to avoid impacts by firening into walls	
at a shallow angle)	45
ArActionBumpers (Action to deal with if the bumpers trigger)	47
ArActionConstantVelocity (Action for going straight at a constant	
velocity)	49
ArActionDesired (Class used to say what movement is desired)	51
ArActionDesiredChannel (Class used by ArActionDesired	
(p. 51) for each channel, internal)	57
ArActionGoto (This action goes to a given ArPose (p. 285) very	
$\mathrm{naively}) \ \ldots \ldots \ldots \ldots \ldots \ldots \ldots$	58
ArActionGroup (Class for groups of actions to accomplish one thing)	60
ArActionGroupInput (Input to drive the robot)	63
ArActionGroupStop (Stop the robot)	64
ArActionGroupTeleop (Teleop the robot)	65
ArActionGroupUnguardedTeleop (Teleop the robot in an un-	
guarded and unsafe manner)	66
ArActionGroupWander (Has the robot wander)	67
ArActionInput (Action for stopping the robot)	68
ArActionJoydrive (This action will use the joystick for input to drive	
the robot)	70
ArActionKeydrive (This action will use the keyboard arrow keys for	
input to drive the robot)	74
- /	

ArActionLimiterBackwards (Action to limit the backwards motion
of the robot) $\dots \dots \dots$
ArActionLimiterForwards (Action to limit the forwards motion of
the robot) $\dots \dots \dots$
ArActionLimiterTableSensor (Action to limit speed based on
whether there the table-sensors see anything) 8
ArActionStallRecover (Action to recover from a stall) 83
ArActionStop (Action for stopping the robot) 85
ArActionTurn (Action to turn when the behaviors with more priority
have limited the speed)
ArACTS_1_2 (Driver for ACTS)
ArACTSBlob (A class for the acts blob)
ArAMPTU (Driver for the AMPUT)
ArAMPTUCommands (A class with the commands for the AMPTU) 99
ArAMPTUPacket (A class for for making commands to send to the
AMPTU)
ArArg (Argument class, mostly for actions, could be used for other
things)
ArArgumentBuilder (This class is to build arguments for things
that require argc and argv)
ArArgumentParser (Class for parsing arguments) 107
ArASyncTask (Asynchronous task (runs in its own thread)) 110
ArBasePacket (Base packet class)
ArCommands (A class with an enum of the commands that can be
sent to the robot)
ArCondition (Threading condition wrapper class)
ArDeviceConnection (Base class for device connections) 124
ArDPPTU (Driver for the DPPTU)
ArDPPTUCommands (A class with the commands for the DPPTU) 136
ArDPPTUPacket (A class for for making commands to send to the
DPPTU)
ArFunctor (Base class for functors)
ArFunctor1 < P1 > (Base class for functors with 1 parameter) 142
ArFunctor1C< T, P1 > (Functor for a member function with 1
parameter)
ArFunctor2 < P1 , P2 > (Base class for functors with 2 parameters) 148
ArFunctor2C < T, P1, P2 > (Functor for a member function with
2 parameters)
ArFunctor3 < P1 , P2 , P3 > (Base class for functors with 3 param-
eters)
ArFunctor3C< T, P1, P2, P3 > (Functor for a member function
with 3 parameters)
ArFunctorASyncTask (This is like ArASyncTask (p. 110), but in-
stead of runThread it uses a functor to run) 164
$\mathbf{ArFunctorC} < \mathbf{T} > (\text{Functor for a member function}) \dots \dots$
ArGlobalFunctor (Functor for a global function with no parameters) 168
` '

ArGlobalFunctor1< P1 > (Functor for a global function with 1
parameter)
ArGlobalFunctor2 < P1, P2 > (Functor for a global function with
2 parameters)
ArGlobalFunctor3 < P1, P2, P3 > (Functor for a global function
with 3 parameters)
ArGlobalRetFunctor< Ret > (Functor for a global function with
return value)
ArGlobalRetFunctor1 < Ret, P1 > (Functor for a global function
with 1 parameter and return value)
ArGlobalRetFunctor2 < Ret, P1, P2 > (Functor for a global func-
tion with 2 parameters and return value) 187
ArGlobalRetFunctor3< Ret, P1, P2, P3 > (Functor for a global
function with 2 parameters and return value) 191
ArGripper (A class of convenience functions for using the gripper) . 196
ArGripperCommands (A class with an enum of the commands for
the gripper)
Aria (This class performs global initialization and deinitialization) 205
ArInterpolation
ArIrrfDevice (A class for connecting to a PB-9 and managing the
resulting data)
ArJoyHandler (Interfaces to a joystick)
ArKeyHandler (This class will read input from the keyboard) 220
ArLine (This is the class for a line to do some geometric manipulation) 224
ArLineSegment (This is the class for a line segment to do some
geometric manipulation)
ArListPos (Has enum for position in list)
ArLog (Logging utility class)
ArLogFileConnection (For connecting through a log file) 233
ArMath (This class has static members to do common math operations) 238
ArMode (A class for different modes, mostly as related to keyboard
input)
ArModeCamera (Mode for controlling the gripper)
ArModeGripper (Mode for controlling the gripper)
ArModeSonar (Mode for displaying the sonar)
ArModeTeleop (Mode for teleoping the robot with joystick + key-
board)
ArModeUnguardedTeleop (Mode for teleoping the robot with joy-
stick + keyboard)
ArModeWander (Mode for wandering around)
,
ArModule (Dynamicly loaded module base class, read warning in more) 260
ArModule (Dynamicly loaded module base class, read warning in more) 260 ArModuleLoader (Dynamic ArModule (p. 260) loader) 263
ArModule (Dynamicly loaded module base class, read warning in more) 260 ArModuleLoader (Dynamic ArModule (p. 260) loader) 263 ArMutex (Mutex wrapper class)
ArModule (Dynamicly loaded module base class, read warning in more) 260 ArModuleLoader (Dynamic ArModule (p. 260) loader) 263

ArNetServerConnection (This class holds the information related	
to specific connections)	71
ArP2Arm (Arm Control class)	
ArPose (The class which represents a position)	
ArPoseWithTime (A subclass of pose that also has the time the	,
pose was taken)	20
• /	
ArPref (Preference instance. Used by ArPreferences)	"
ArPriorityResolver ((Default resolver), takes the action list and uses	_
the priority to resolve)	
ArPTZ (Base class which handles the PTZ cameras) 29	
ArRangeBuffer (This class is a buffer that holds ranging information) 30)[
ArRangeDevice (The class for all devices which return range info	
(laser, sonar))	2
ArRangeDeviceThreaded (A range device which can run in its own	
thread)	2(
ArRecurrent Task (Recurrent task (runs in its own thread)) 32	
ArResolver (Resolves a list of actions and returns what to do) 32	
ArRetFunctor< Ret > (Base class for functors with a return value) 32	
ArRetFunctor1< Ret, P1 > (Base class for functors with a return	- 1
value with 1 parameter)) (
	i C
ArRetFunctor1C< Ret, T, P1 > (Functor for a member function	
with return value and 1 parameter)	3(
ArRetFunctor2< Ret, P1, P2 > (Base class for functors with a	
return value with 2 parameters)	34
ArRetFunctor2C< Ret, T, P1, P2 > (Functor for a member func-	
tion with return value and 2 parameters) 33	36
ArRetFunctor3 < Ret, P1, P2, P3 > (Base class for functors with	
a return value with 3 parameters)	12
ArRetFunctor3C< Ret, T, P1, P2, P3 > (Functor for a member	
function with return value and 3 parameters)	1.5
$\mathbf{ArRetFunctorC} < \mathbf{Ret}, \ \mathbf{T} > (\mathbf{Functor} \ \mathbf{for} \ \mathbf{a} \ \mathbf{member} \ \mathbf{function} \ \mathbf{with}$	
return value)	5
ArRobot (THE important class)	
) (
ArRobotPacket (Represents the packets sent to the robot as well as	١,
those received from it)	JC
ArRobotPacketReceiver (Given a device connection it receives	
packets from the robot through it))8
ArRobotPacketSender (Given a device connection this sends com-	
mands through it to the robot) 41	1
ArRobotParams (Contains the robot parameters, according to the	
parameter file)	Ę
ArSectors (A class for keeping track of if a complete revolution has	
been attained)	8
ArSensorReading (A class to hold a sensor reading, should be one	
instance per sensor)	ļ
ArSerialConnection (For connecting to devices through a serial port) 42	
	-

ArSick (The sick driver)	431
ArSickLogger (This class can be used to create log files for the laser	
$\mathrm{mapper}) \ \ldots \ $	444
ArSickPacket (Represents the packets sent to the sick as well as those	
received from it) \dots	447
ArSickPacketReceiver (Given a device connection it receives pack-	
ets from the sick through it)	450
ArSignalHandler (Signal handling class)	453
ArSimpleConnector (This class simplifies connecting to the robot	
$and/or laser) \dots \dots \dots \dots \dots$	459
ArSocket (Socket communication wrapper)	461
ArSonarDevice (A class for keeping track of sonar)	468
ArSonyPacket (A class for for making commands to send to the sony))470
ArSonyPTZ (A class to use the sony pan tilt zoom unit)	472
ArSyncTask (Class used internally to manage the functions that are	
called every cycle)	475
ArTaskState (Class with the different states a task can be in)	480
ArTcpConnection (For connectiong to a device through a socket) .	481
ArThread (POSIX/WIN32 thread wrapper class)	487
ArTime (A class for time readings)	491
ArTransform (A class to handle transforms between different coor-	
dinates) \dots	493
ArTypes (Contains platform independent sized variable types)	496
ArUtil (This class has utility functions)	497
ArVCC4 (Driver for the VCC4)	506
ArVCC4Commands (A class with the commands for the VCC4)	512
ArVCC4Packet (A class for for making commands to send to the	
VCC4)	514
P2ArmJoint (P2 Arm joint info)	515

Chapter 4

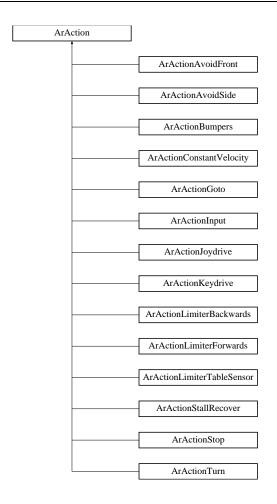
Aria Class Documentation

4.1 ArAction Class Reference

Action class, what typically makes the robot move.

#include <ArAction.h>

Inheritance diagram for ArAction::



Public Methods

- **ArAction** (const char *name, const char *description="") *Constructor.*
- virtual ~**ArAction** ()

 Desructor.
- virtual bool **isActive** (void) const Finds out whether the action is active or not.
- virtual void **activate** (void)

 Activate the action.

• virtual void deactivate (void)

Deactivate the action.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** current-Desired)=0

Fires the action, returning what the action wants to do.

virtual void setRobot (ArRobot *robot)

Sets the robot this action is driving.

• virtual int **getNumArgs** (void) const

Find the number of arguments this action takes.

 \bullet virtual const $\mathbf{ArArg} * \mathbf{getArg}$ (int number) const

Gets the numbered argument.

• virtual **ArArg** * **getArg** (int number)

Gets the numbered argument.

• virtual const char * **getName** (void) const

Gets the name of the action.

• virtual const char * **getDescription** (void) const

Gets the long description of the action.

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

• virtual void log (bool verbose=true) const

ArLog::log (p. 232) s the actions stats.

Protected Methods

• void **setNextArgument** (**ArArg** const & arg)

Sets the argument type for the next argument (only use in constructor).

4.1.1 Detailed Description

Action class, what typically makes the robot move.

4.1.2 Member Function Documentation

4.1.2.1 virtual ArActionDesired* ArAction::fire (ArActionDesired currentDesired) [pure virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented in ArActionAvoidFront (p. 44), ArActionAvoidSide (p. 46), ArActionBumpers (p. 48), ArActionConstantVelocity (p. 50), ArActionGoto (p. 59), ArActionInput (p. 69), ArActionJoydrive (p. 72), ArActionKeydrive (p. 75), ArActionLimiterBackwards (p. 78), ArActionLimiterForwards (p. 80), ArActionLimiterTableSensor (p. 82), ArActionStallRecover (p. 84), ArActionStop (p. 86), and ArActionTurn (p. 88).

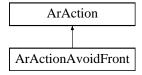
- ArAction.h
- ArAction.cpp

4.2 ArActionAvoidFront Class Reference

This action does obstacle avoidance, controlling both trans and rot.

#include <ArActionAvoidFront.h>

Inheritance diagram for ArActionAvoidFront::



Public Methods

• ArActionAvoidFront (const char *name="avoid front obstacles", double obstacleDistance=450, double avoidVelocity=200, double turn-Amount=15, bool useTableIRIfAvail=true)

Constructor.

• virtual ~ArActionAvoidFront ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

Fires the action, returning what the action wants to do.

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.2.1 Detailed Description

This action does obstacle avoidance, controlling both trans and rot.

This action uses whatever available range device have been added to the robot to avoid obstacles. See the ArActionAvoidFront constructor documentation to see the parameters it takes.

Also note that this action does something most others don't, which is to check for a specific piece of hardware. This is the tableSensingIR. If this is set up in the parameters for the robot, it will use DigIn0 and DigIn1, where the tableSensingIRs are connected. Note that if you make useTableIRIfAvail false in the constructor it'll ignore these. Whether the action thinks the robot has them or not depends on the value of tableSensingIR in the parameter file for that robot.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ArActionAvoidFront::ArActionAvoidFront (const char * name = "avoid front obstacles", double obstacleDistance = 450, double avoidVelocity = 200, double turnAmount = 15, bool useTableIRIfAvail = true)

Constructor.

Parameters:

name the name of the action

obstacleDistance distance at which to turn. (mm)

avoid Velocity Speed at which to go while avoiding an obstacle. (mm/sec)

turnAmount Degrees to turn relative to current heading while avoiding obstacle (deg)

use Table IRIfAvail Whether to use the table sensing IR if they are available

4.2.3 Member Function Documentation

4.2.3.1 ArActionDesired * ArActionAvoidFront::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

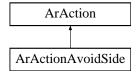
- ArActionAvoidFront.h
- ArActionAvoidFront.cpp

4.3 ArActionAvoidSide Class Reference

Action to avoid impacts by firening into walls at a shallow angle.

#include <ArActionAvoidSide.h>

Inheritance diagram for ArActionAvoidSide::



Public Methods

• ArActionAvoidSide (const char *name="Avoid side", double obstacle-Distance=300, double turnAmount=5)

Constructor.

• virtual ~ArActionAvoidSide ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

Fires the action, returning what the action wants to do.

• virtual **ArActionDesired** * **getDesired** (void)

Gets what this action wants to do (for display purposes).

4.3.1 Detailed Description

Action to avoid impacts by firening into walls at a shallow angle.

This action watches the sensors to see if it is close to firening into a wall at a shallow enough angle that other avoidance may not avoid.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ArActionAvoidSide::ArActionAvoidSide (const char * name = "Avoid side", double obstacleDistance = 300, double turnAmount = 5)

Constructor.

Parameters:

name name of the action
obstacleDistance distance at which to start avoiding (mm)
turnAmount degrees at which to turn (deg)

4.3.3 Member Function Documentation

4.3.3.1 ArActionDesired * ArActionAvoidSide::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from ArAction (p. 42).

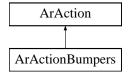
- ArActionAvoidSide.h
- ArActionAvoidSide.cpp

4.4 ArActionBumpers Class Reference

Action to deal with if the bumpers trigger.

#include <ArActionBumpers.h>

Inheritance diagram for ArActionBumpers::



Public Methods

• ArActionBumpers (const char *name="bumpers", double back-OffSpeed=100, int backOffTime=2500, int turnTime=500, bool set-Maximums=false)

Constructor.

• virtual ~ArActionBumpers ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

Fires the action, returning what the action wants to do.

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.4.1 Detailed Description

Action to deal with if the bumpers trigger.

This class basically responds to the bumpers the robot has, what the activity things the robot has is decided by the param file. If the robot is going forwards and bumps into something with the front bumpers, it will back up and turn. If the robot is going backwards and bumps into something with the rear bumpers then the robot will move forward and turn.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ArActionBumpers::ArActionBumpers (const char * name = "bumpers", double backOffSpeed = 100, int backOffTime = 2500, int turnTime = 500, bool setMaximums = false)

Constructor.

Parameters:

name name of the action
backOffSpeed speed at which to back away (mm/sec)
backOffTime number of msec to back up for (msec)
turnTime number of msec to alow for turn (msec)

4.4.3 Member Function Documentation

4.4.3.1 ArActionDesired * ArActionBumpers::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from ArAction (p. 42).

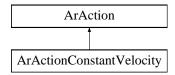
- ArActionBumpers.h
- ArActionBumpers.cpp

4.5 ArActionConstantVelocity Class Reference

Action for going straight at a constant velocity.

#include <ArActionConstantVelocity.h>

Inheritance diagram for ArActionConstantVelocity::



Public Methods

• ArActionConstantVelocity (const char *name="Constant Velocity", double velocity=400)

Constructor.

• virtual ~ArActionConstantVelocity ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

Fires the action, returning what the action wants to do.

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.5.1 Detailed Description

Action for going straight at a constant velocity.

This action simply goes straight at a constant velocity.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ArActionConstantVelocity::ArActionConstantVelocity (const char * name = "Constant Velocity", double velocity = 400)

Constructor.

Parameters:

name name of the action
velocity velocity to travel at (mm/sec)

4.5.3 Member Function Documentation

4.5.3.1 ArActionDesired * ArActionConstantVelocity::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

- $\bullet \ \ ArActionConstantVelocity.h$
- ArActionConstantVelocity.cpp

4.6 ArActionDesired Class Reference

Class used to say what movement is desired.

#include <ArActionDesired.h>

Public Methods

• ArActionDesired ()

Constructor.

• virtual ~ArActionDesired ()

Destructor.

- virtual void **setVel** (double vel, double strength=MAX_STRENGTH)

 Sets the velocity (mm/sec) and strength.
- virtual void **setDeltaHeading** (double deltaHeading, double strength=MAX_STRENGTH)

Sets the delta heading (deg) and strength.

• virtual void **setHeading** (double heading, double strength=MAX_-STRENGTH)

Sets the absolute heading (deg).

ullet virtual void $\mathbf{setMaxVel}$ (double \mathbf{maxVel} , double $\mathbf{strength} = \mathbf{MAX} - \mathbf{STRENGTH}$)

Sets the maximum velocity (+mm/sec) and strength.

• virtual void **setMaxNegVel** (double maxVel, double strength=MAX_-STRENGTH)

Sets the maximum velocity for going backwards (-mm/sec) and strength.

• virtual void **setMaxRotVel** (double maxVel, double strength=MAX_-STRENGTH)

Sets the maximum rotational velocity (deg/sec) and strength.

• virtual void **reset** (void)

Resets the strengths to 0.

• virtual double **getVel** (void)

Gets the translational velocity desired (mm/sec).

• virtual double **getVelStrength** (void)

Gets the strength of the translational velocity desired.

• virtual double **getHeading** (void)

Gets the heading desired (deg).

• virtual double **getHeadingStrength** (void)

Gets the strength of the heading desired.

• virtual double **getDeltaHeading** (void)

Gets the delta heading desired (deg).

• virtual double **getDeltaHeadingStrength** (void)

Gets the strength of the delta heading desired.

• virtual double **getMaxVel** (void)

Gets the desired maximum velocity (mm/sec).

• virtual double **getMaxVelStrength** (void)

Gets the maximum velocity strength.

• virtual double **getMaxNegVel** (void)

Gets the desired maximum negative velocity (-mm/sec).

• virtual double **getMaxNegVelStrength** (void)

Gets the desired maximum negative velocity strength.

• virtual double **getMaxRotVel** (void)

Gets the maximum rotational velocity.

• virtual double **getMaxRotVelStrength** (void)

Gets the maximum rotational velocity strength.

• virtual void merge (ArActionDesired *actDesired)

Merges the given ArActionDesired into this one (this one has precedence), internal.

• virtual void **startAverage** (void)

Starts the process of avereraging together different desireds.

• virtual void addAverage (ArActionDesired *actDesired)

Adds another actionDesired into the mix to average.

• virtual void **endAverage** (void)

Ends the process of avereraging together different desireds.

• virtual void accountForRobotHeading (double robotHeading)

Accounts for robot heading, mostly internal.

• ArActionDesiredChannel * getVelDesiredChannel (void)

Accessor for the channel structor for merge, internal.

• ArActionDesiredChannel * getDeltaHeadingDesiredChannel (void)

Accessor for the channel structor for merge, internal.

• ArActionDesiredChannel * getMaxVelDesiredChannel (void)

Accessor for the channel structor for merge, internal.

 \bullet ArActionDesiredChannel * getMaxNegVelDesiredChannel (void)

Accessor for the channel structor for merge, internal.

• ArActionDesiredChannel * getMaxRotVelDesiredChannel (void)

Accessor for the channel structor for merge, internal.

4.6.1 Detailed Description

Class used to say what movement is desired.

This class is use by actions to report what they want to want to do (hence the name).

The way it works, is that translational (front/back) and rotational (right/left) are seperate. Translational movement uses velocity, while rotational movement uses change in heading from current heading. Translational and rotational each have their own strength value. Both translational and rotational movement have maximum velocities as well, that also have their own strengths.

The strength value reflects how strongly an action wants to do the chosen movement command, the resolver (ArResolver (p. 325)) will combine these strengths and figure out what to do based on them.

For all strength values there is a total of 1.0 strength to be had. The range for strength is from 0 to 1. This is simply a convention that ARIA uses by default, if you don't like it, you can override this class and make an **ArResolver** (p. 325).

4.6.2 Member Function Documentation

4.6.2.1 virtual void ArActionDesired::accountForRobotHeading (double robotHeading) [inline, virtual]

Accounts for robot heading, mostly internal.

This accounts for the robots heading, and transforms the set heading on this actionDesired into a delta heading so it can be merged and averaged and the like

Parameters:

robotHeading the heading the real actual robot is at now

4.6.2.2 virtual void ArActionDesired::addAverage (ArActionDesired * actDesired) [inline, virtual]

Adds another actionDesired into the mix to average.

For a description of how to use this, see startAverage.

Parameters:

actDesired the actionDesired to add into the average

4.6.2.3 virtual void ArActionDesired::endAverage (void) [inline, virtual]

Ends the process of avereraging together different desireds.

For a description of how to use this, see startAverage.

4.6.2.4 virtual void ArActionDesired::merge (ArActionDesired * actDesired) [inline, virtual]

Merges the given ArActionDesired into this one (this one has precedence), internal

This merges in the two different action values, accountForRobotHeading MUST be done before this is called (on both actions), since this merges their delta headings, and the deltas can't be known unless the account for angle is done.

Parameters:

actDesired the actionDesired to merge with this one

4.6.2.5 virtual void ArActionDesired::setDeltaHeading (double deltaHeading, double strength = MAX_STRENGTH) [inline, virtual]

Sets the delta heading (deg) and strength.

Parameters:

deltaHeading desired change in heading (deg)
strength strength given to this, defaults to MAX_STRENGTH (1.0)

4.6.2.6 virtual void ArActionDesired::setHeading (double *heading*, double *strength* = MAX_STRENGTH) [inline, virtual]

Sets the absolute heading (deg).

This is a way to set the heading instead of using a delta, there is no get for this, because accountForRobotHeading MUST be called (this should be called by all resolvers, but if you want to call it you can, thats fine).

Parameters:

heading desired heading (deg) strength strength given to this, defaults to MAX_STRENGTH (1.0)

4.6.2.7 virtual void ArActionDesired::setMaxNegVel (double maxVel, double $strength = MAX_STRENGTH$) [inline, virtual]

Sets the maximum velocity for going backwards (-mm/sec) and strength.

Parameters:

maxVel desired maximum velocity for going backwards (-mm/sec) strength strength given to this, defaults to MAX_STRENGTH (1.0)

4.6.2.8 virtual void ArActionDesired::setMaxRotVel (double maxVel, double $strength = MAX_STRENGTH$) [inline, virtual]

Sets the maximum rotational velocity (deg/sec) and strength.

Parameters:

maxVel desired maximum rotational velocity (deg/sec) strength strength given to this, defaults to MAX_STRENGTH (1.0)

4.6.2.9 virtual void ArActionDesired::setMaxVel (double maxVel, double strength = MAX_STRENGTH) [inline, virtual]

Sets the maximum velocity (+mm/sec) and strength.

Parameters:

```
maxVel desired maximum velocity (+mm/sec)
strength strength given to this, defaults to MAX_STRENGTH (1.0)
```

4.6.2.10 virtual void ArActionDesired::setVel (double vel, double $strength = MAX_STRENGTH$) [inline, virtual]

Sets the velocity (mm/sec) and strength.

Parameters:

```
vel desired vel (mm/sec)strength strength given to this, defaults to MAX_STRENGTH (1.0)
```

4.6.2.11 virtual void ArActionDesired::startAverage (void) [inline, virtual]

Starts the process of avereraging together different desireds.

There is a three step process for averaging actionDesireds together, first start-Average must be done to set up the process, then addAverage must be done with each average that is desired, then finally endAverage should be used, after that is done then the normal process of getting the results out should be done.

- ArActionDesired.h
- ArActionDesired.cpp

4.7 ArActionDesiredChannel Class Reference

Class used by **ArActionDesired** (p. 51) for each channel, internal.

#include <ArActionDesired.h>

4.7.1 Detailed Description

Class used by **ArActionDesired** (p. 51) for each channel, internal.

4.7.2 Member Data Documentation

4.7.2.1 const double ArActionDesiredChannel::MAX_STRENGTH [static]

Initial value:

ArActionDesired::MAX_STRENGTH

4.7.2.2 const double ArActionDesiredChannel::MIN_STRENGTH [static]

Initial value:

ArActionDesired::MIN_STRENGTH

4.7.2.3 const double ArActionDesiredChannel::NO_STRENGTH [static]

Initial value:

ArActionDesired::NO_STRENGTH

- ArActionDesired.h
- ArActionDesired.cpp

4.8 ArActionGoto Class Reference

This action goes to a given **ArPose** (p. 285) very naively.

#include <ArActionGoto.h>

Inheritance diagram for ArActionGoto::



Public Methods

- bool haveAchievedGoal (void)
 Sees if the goal has been achieved.
- void cancelGoal (void)

 Cancels the goal the robot has.
- void **setGoal** (**ArPose** goal)

 Sets a new goal and sets the action to go there.
- ArPose getGoal (void)

Gets the goal the action has.

- void **setCloseDist** (double closeDist)

 Set the distance which is close enough to the goal (mm);.
- double **getCloseDist** (void)

 Gets the distance which is close enough to the goal (mm).
- void **setSpeed** (double speed)

 Sets the speed the action will travel to the goal at (mm/sec).
- double **getSpeed** (void)

 Gets the speed the action will travel to the goal at (mm/sec).
- virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

 Fires the action, returning what the action wants to do.

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.8.1 Detailed Description

This action goes to a given ArPose (p. 285) very naively.

This action naively drives straight towards a given **ArPose** (p. 285)... the action stops when it gets closeDist away... it travels to the point at speed mm/sec.

You can give it a new goal with setGoal, cancel its movement with cancelGoal, and see if it got there with haveAchievedGoal.

This doesn't avoid obstacles or anything, you could have an avoid routine at a higher priority to avoid on the way there... but for real and intelligent looking navigation you should use something like Saphira's Gradient navigation.

4.8.2 Member Function Documentation

4.8.2.1 ArActionDesired * ArActionGoto::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

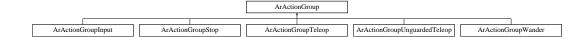
- ArActionGoto.h
- ArActionGoto.cpp

4.9 ArActionGroup Class Reference

Class for groups of actions to accomplish one thing.

#include <ArActionGroup.h>

Inheritance diagram for ArActionGroup::



Public Methods

• ArActionGroup (ArRobot *robot)

Constructor.

• virtual ~ArActionGroup ()

Destructor, it also deletes the actions in its group.

- virtual void addAction (ArAction *action, int priority)
 Adds the action to the robot this group uses with the given priority.
- virtual void remAction (ArAction *action)
 Removes the action from the robot this group uses.
- virtual void **activate** (void)

 Activates all the actions in this group.
- virtual void activateExclusive (void)
 Activates all the actions in this group and deactivates all others.
- virtual void **deactivate** (void)

 Deactivates all the actions in this group.
- virtual void removeActions (void)
 Removes all the actions in this group from the robot.
- virtual std::list< ArAction *> * getActionList (void)

 Gets the action list (use this to delete actions after doing removeActions).

4.9.1 Detailed Description

Class for groups of actions to accomplish one thing.

This class is used to have a group of ArActions and turn them on and off in aggregate... this is so that you can say have a group of like 5 behaviors for teleop or wander, and just turn 'em all on and off at once. Note that the destructor by default will delete the actions added to the group, this is controlled with a flag to the constructor though, so you can have it how you want.... this is nice though so you can just do addAction(new ArActionWhatever(blah, blah, blah), 90); and not worry about the deletion (since the destructor will do it), just delete the group... if for some reason (I'd advise against it) you are using one action in multiple groups, don't use this feature, ie pass in false to the constructor for it or you'll wind up with a crash when the action is deleted by both groups (again, you should probably only have an action in one group).

4.9.2 Constructor & Destructor Documentation

4.9.2.1 ArActionGroup::ArActionGroup (ArRobot * robot)

Constructor.

@param robot The robot that this action group is attached to

@param deleteActionsOnDestruction if this is true then when the destructor is called the actions that this group has will be deleted

4.9.3 Member Function Documentation

4.9.3.1 void ArActionGroup::addAction (ArAction * action, int priority) [virtual]

Adds the action to the robot this group uses with the given priority.

@param action the action to add to the robot @param priority the priority to give the action @see **ArRobot::addAction** (p. 371)

4.9.3.2 void ArActionGroup::remAction (ArAction * action) [virtual]

Removes the action from the robot this group uses.

@param action the action to remove from the robot @see **ArRobot::rem-Action** (p. 393)

- $\bullet \ \, ArActionGroup.h$
- $\bullet \ \, ArActionGroup.cpp$

4.10 ArActionGroupInput Class Reference

Input to drive the robot.

#include <ArActionGroups.h>

Inheritance diagram for ArActionGroupInput::



4.10.1 Detailed Description

Input to drive the robot.

This class is just useful for teleoping the robot under your own joystick and keyboard control... Note that you the predefined ArActionGroups in ARIA are made only to be used exclusively... they won't combine.

- ArActionGroups.h
- ArActionGroups.cpp

4.11 ArActionGroupStop Class Reference

Stop the robot.

#include <ArActionGroups.h>

Inheritance diagram for ArActionGroupStop::



4.11.1 Detailed Description

Stop the robot.

This class is just useful for having the robot stopped... Note that you the predefined ArActionGroups in ARIA are made only to be used exclusively... they won't combine.

- ArActionGroups.h
- $\bullet \ \, {\rm ArActionGroups.cpp}$

4.12 ArActionGroupTeleop Class Reference

Teleop the robot.

#include <ArActionGroups.h>

Inheritance diagram for ArActionGroupTeleop::



4.12.1 Detailed Description

Teleop the robot.

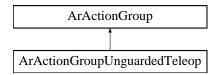
This class is just useful for teleoping the robot and having these actions read the joystick and keyboard... Note that you the predefined ArActionGroups in ARIA are made only to be used exclusively... they won't combine.

- ArActionGroups.h
- ArActionGroups.cpp

Teleop the robot in an unguarded and unsafe manner.

#include <ArActionGroups.h>

 $Inheritance\ diagram\ for\ ArAction Group Unguarded Teleop::$



4.13.1 Detailed Description

Teleop the robot in an unguarded and unsafe manner.

This class is just useful for teleoping the robot in an unguarded and unsafe manner and having these actions read the joystick and keyboard... Note that you the predefined ArActionGroups in ARIA are made only to be used exclusively... they won't combine.

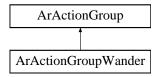
- ArActionGroups.h
- ArActionGroups.cpp

4.14 ArActionGroupWander Class Reference

Has the robot wander.

#include <ArActionGroups.h>

Inheritance diagram for ArActionGroupWander::



4.14.1 Detailed Description

Has the robot wander.

This class is useful for having the robot wander... Note that you the predefined ArActionGroups in ARIA are made only to be used exclusively... they won't combine.

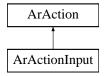
- ArActionGroups.h
- ArActionGroups.cpp

4.15 ArActionInput Class Reference

Action for stopping the robot.

#include <ArActionInput.h>

Inheritance diagram for ArActionInput::



Public Methods

- **ArActionInput** (const char *name="Input")

 Constructor.
- virtual ~**ArActionInput** ()

 Destructor.
- void setVel (double vel)

Set velocity (cancels deltaVel).

- void **deltaVel** (double delta)

 Increment/decrement the velocity (cancels setVel).
- void **deltaHeading** (double delta)

 Increment/decrement the heading.
- void deltaHeadingFromCurrent (double delta)
 Increment/decrement the heading from current.
- virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

 Fires the action, returning what the action wants to do.
- virtual ArActionDesired * getDesired (void)
 Gets what this action wants to do (for display purposes).
- void activate (void)

 Activate the action.

4.15.1 Detailed Description

Action for stopping the robot.

This action simply sets the robot to a 0 velocity and a deltaHeading of 0.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 ArActionInput::ArActionInput (const char * name = "Input")

Constructor.

Parameters:

name name of the action

4.15.3 Member Function Documentation

4.15.3.1 ArActionDesired * ArActionInput::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

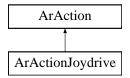
- ArActionInput.h
- ArActionInput.cpp

4.16 ArActionJoydrive Class Reference

This action will use the joystick for input to drive the robot.

#include <ArActionJoydrive.h>

Inheritance diagram for ArActionJoydrive::



Public Methods

• ArActionJoydrive (const char *name="joydrive", double trans-VelMax=400, double turnAmountMax=15, bool stopIfNoButton-Pressed=true, bool useOSCalForJoystick=true)

Constructor.

 $\bullet \ \, {\rm virtual} \sim \! \mathbf{ArActionJoydrive} \,\, () \\$

Destructor.

 $\bullet \ \, {\rm virtual} \ \, \mathbf{ArActionDesired} * \mathbf{fire} \ \, (\mathbf{ArActionDesired} \ \, \mathrm{currentDesired})$

Fires the action, returning what the action wants to do.

• bool joystickInited (void)

Whether the joystick is initalized or not.

- void **setSpeeds** (double transVelMax, double turnAmountMax) Set Speeds.
- void **setStopIfNoButtonPressed** (bool stopIfNoButtonPressed)

 Set if we'll stop if no button is pressed, otherwise just do nothing.
- bool getStopIfNoButtonPressed (void)
 - Get if we'll stop if no button is pressed, otherwise just do nothing.
- void **setThrottleParams** (int lowSpeed, int highSpeed)

 Sets the params on the throttle (throttle unused unless you call this).
- void setUseOSCal (bool useOSCal)

Sets whether to use OSCalibration the joystick or not.

• bool getUseOSCal (void)

Gets whether OSCalibration is being used for the joystick or not.

• ArJoyHandler * getJoyHandler (void)

Gets the joyHandler.

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.16.1 Detailed Description

This action will use the joystick for input to drive the robot.

This class creates its own **ArJoyHandler** (p. 214) to get input from the joystick. Then it will scale the speed between 0 and the given max for velocity and turning, up and down on the joystick go forwards/backwards while right and left go right and left. You must press in one of the two joystick buttons for the class to pay attention to the joystick.

NOTE: The joystick does not save calibration information, so you must calibrate the joystick before each time you use it. To do this, press the button for at least a half a second while the joystick is in the middle. Then let go of the button and hold the joystick in the upper left for at least a half second and then in the lower right corner for at least a half second.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 ArActionJoydrive::ArActionJoydrive (const char * name = "joydrive", double transVelMax = 400, double turnAmountMax = 15, bool stopIfNoButtonPressed = true, bool useOSCalForJoystick = true)

Constructor.

This action is for driving around the robot with a joystick, you must hold in a button on the joystick and then lean the joystick over to have it drive. You need to calibrate the joystick for it to work right, for details about this see **ArJoyHandler** (p. 214).

Parameters:

name the name of this action

- trans VelMax the maximum velocity the joydrive action will go, it reachs this when the joystick is all the way forwards
- turnAmountMax the maximum amount the joydrive action will turn, it reachs this when the joystick is all the way forwards
- stopIfNoButtonPressed if this is true and there is a joystick and no button is pressed, the action will have the robot stop... otherwise it'll do nothing (letting lower priority actions fire)

See also:

ArJoyHandler::setUseOSCal (p. 219)

4.16.3 Member Function Documentation

4.16.3.1 ArActionDesired * ArActionJoydrive::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

4.16.3.2 bool ArActionJoydrive::getUseOSCal (void)

Gets whether OSCalibration is being used for the joystick or not.

See also:

ArJoyHandler::getUseOSCal (p. 218)

4.16.3.3 void ArActionJoydrive::setUseOSCal (bool useOSCal)

Sets whether to use OSCalibration the joystick or not.

See also:

ArJoyHandler::setUseOSCal (p. 219)

- $\bullet \ \, Ar Action Joy drive.h$
- $\bullet \ \ ArActionJoydrive.cpp$

4.17 ArActionKeydrive Class Reference

This action will use the keyboard arrow keys for input to drive the robot.

#include <ArActionKeydrive.h>

Inheritance diagram for ArActionKeydrive::



Public Methods

• ArActionKeydrive (const char *name="keydrive", double transVel-Max=400, double turnAmountMax=24, double velIncrement=25, double turnIncrement=8)

Constructor.

 \bullet virtual \sim **ArActionKeydrive** ()

Destructor.

 $\bullet \ \, {\rm virtual} \ \, \mathbf{ArActionDesired} * \mathbf{fire} \ \, (\mathbf{ArActionDesired} \ \, \mathrm{currentDesired})$

Fires the action, returning what the action wants to do.

• void **setSpeeds** (double transVelMax, double turnAmountMax)

For setting the maximum speeds.

• void **setIncrements** (double velIncrement, double turnIncrement)

For setting the increment amounts.

• virtual **ArActionDesired** * **getDesired** (void)

Gets what this action wants to do (for display purposes).

• virtual void **setRobot** (**ArRobot** *robot)

Sets the robot this action is driving.

• virtual void **activate** (void)

Activate the action.

• virtual void **deactivate** (void)

Deactivate the action.

• void takeKeys (void)

Takes the keys this action wants to use to drive.

• void **giveUpKeys** (void)

Gives up the keys this action wants to use to drive.

• void **up** (void)

Internal, callback for up arrow.

• void **down** (void)

Internal, callback for down arrow.

• void **left** (void)

Internal, callback for left arrow.

• void **right** (void)

Internal, callback for right arrow.

• void **space** (void)

Internal, callback for space key.

4.17.1 Detailed Description

This action will use the keyboard arrow keys for input to drive the robot.

4.17.2 Member Function Documentation

4.17.2.1 ArActionDesired * ArActionKeydrive::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

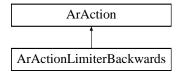
- $\bullet \ \, Ar Action Keydrive.h$
- $\bullet \ \, {\rm ArActionKeydrive.cpp}$

4.18 ArActionLimiterBackwards Class Reference

Action to limit the backwards motion of the robot.

#include <ArActionLimiterBackwards.h>

Inheritance diagram for ArActionLimiterBackwards::



Public Methods

• ArActionLimiterBackwards (const char *name="speed limiter", double stopDistance=-250, double slowDistance=-600, double max-BackwardsSpeed=-250)

Constructor.

• virtual ~ArActionLimiterBackwards ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

Fires the action, returning what the action wants to do.

 \bullet virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.18.1 Detailed Description

Action to limit the backwards motion of the robot.

This class limits the backwards motion of the robot according to the parameters given.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 ArActionLimiterBackwards::ArActionLimiterBackwards (const char * name = "speed limiter", double stopDistance = -250, double slowDistance = -600, double maxBackwardsSpeed = -250)

Constructor.

Parameters:

name name of the action
stopDistance distance at which to stop (mm)
slowDistance distance at which to slow down (mm)
maxBackwardsSpeed maximum backwards speed, speed allowed scales from this to 0 at the stop distance (mm/sec)

4.18.3 Member Function Documentation

4.18.3.1 ArActionDesired * ArActionLimiterBackwards::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

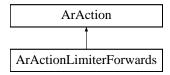
- ArActionLimiterBackwards.h
- $\bullet \ \ Ar Action Limiter Backwards. cpp$

4.19 ArActionLimiterForwards Class Reference

Action to limit the forwards motion of the robot.

#include <ArActionLimiterForwards.h>

Inheritance diagram for ArActionLimiterForwards::



Public Methods

• ArActionLimiterForwards (const char *name="speed limiter", double stopDistance=250, double slowDistance=600, double slowSpeed=250, double widthRatio=1.5)

Constructor.

• virtual ~ArActionLimiterForwards ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

 $Fires\ the\ action,\ returning\ what\ the\ action\ wants\ to\ do.$

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.19.1 Detailed Description

Action to limit the forwards motion of the robot.

This action uses the sensors to find a maximum speed to travel at

4.19.2 Constructor & Destructor Documentation

4.19.2.1 ArActionLimiterForwards::ArActionLimiterForwards (const char * name = "speed limiter", double stopDistance = 250, double slowDistance = 600, double slowSpeed = 250, double widthRatio = 1.5)

Constructor.

Parameters:

```
name name of the action
stopDistance distance at which to stop (mm)
slowDistance distance at which to slow down (mm)
slowSpeed speed allowed at slowDistance, scales to 0 at slow distance (mm/sec)
```

4.19.3 Member Function Documentation

4.19.3.1 ArActionDesired * ArActionLimiterForwards::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

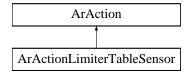
- ArActionLimiterForwards.h
- ArActionLimiterForwards.cpp

4.20 ArActionLimiterTableSensor Class Reference

Action to limit speed based on whether there the table-sensors see anything.

#include <ArActionLimiterTableSensor.h>

Inheritance diagram for ArActionLimiterTableSensor::



Public Methods

• ArActionLimiterTableSensor (const char *name="TableSensor-Limiter")

Constructor.

• virtual ~ArActionLimiterTableSensor ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

Fires the action, returning what the action wants to do.

• virtual **ArActionDesired** * **getDesired** (void)

Gets what this action wants to do (for display purposes).

4.20.1 Detailed Description

Action to limit speed based on whether there the table-sensors see anything.

This action limits speed to 0 if the table-sensors see anything in front of the robot. The action will only work if the robot has table sensors, meaning that the robots parameter file has them listed as true.

4.20.2 Member Function Documentation

4.20.2.1 ArActionDesired * ArActionLimiterTableSensor::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

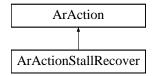
- ArActionLimiterTableSensor.h
- ArActionLimiterTableSensor.cpp

4.21 ArActionStallRecover Class Reference

Action to recover from a stall.

#include <ArActionStallRecover.h>

Inheritance diagram for ArActionStallRecover::



Public Methods

• ArActionStallRecover (const char *name="stall recover", double obstacleDistance=225, int cyclesToMove=50, double speed=150, double degreesToTurn=45)

Constructor.

• virtual \sim ArActionStallRecover ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

 $Fires\ the\ action,\ returning\ what\ the\ action\ wants\ to\ do.$

• virtual ArActionDesired * getDesired (void)

Gets what this action wants to do (for display purposes).

4.21.1 Detailed Description

Action to recover from a stall.

This action tries to recover if one of the wheels has stalled, it has a series of actions it tries in order to get out of the stall.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 AREXPORT ArActionStallRecover::ArActionStallRecover (const char * name = "stall recover", double obstacleDistance = 225, int cyclesToMove = 50, double speed = 150, double degreesToTurn = 45)

Constructor.

Parameters:

name name of the action
obstacleDistance distance at which not to move because of obstacle. (mm)
cyclesToMove number of cycles to move (# of cycles)
speed speed at which to back up or go forward (mm/sec)
degreesToTurn number of degrees to turn (deg)

4.21.3 Member Function Documentation

4.21.3.1 AREXPORT ArActionDesired * ArActionStall-Recover::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

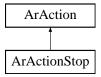
- ArActionStallRecover.h
- ArActionStallRecover.cpp

4.22 ArActionStop Class Reference

Action for stopping the robot.

#include <ArActionStop.h>

Inheritance diagram for ArActionStop::



Public Methods

- **ArActionStop** (const char *name="stop")

 Constructor.
- virtual ~**ArActionStop** ()

 Destructor.
- virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

 Fires the action, returning what the action wants to do.
- virtual **ArActionDesired** * **getDesired** (void)

 Gets what this action wants to do (for display purposes).

4.22.1 Detailed Description

Action for stopping the robot.

This action simply sets the robot to a 0 velocity and a deltaHeading of 0.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 ArActionStop::ArActionStop (const char * name = "stop")

Constructor.

Parameters:

name name of the action

4.22.3 Member Function Documentation

4.22.3.1 ArActionDesired * ArActionStop::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

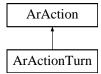
- ArActionStop.h
- ArActionStop.cpp

4.23 ArActionTurn Class Reference

Action to turn when the behaviors with more priority have limited the speed.

#include <ArActionTurn.h>

Inheritance diagram for ArActionTurn::



Public Methods

• ArActionTurn (const char *name="turn", double speedStartTurn=200, double speedFullTurn=100, double turnAmount=15)

Constructor.

• virtual ~ArActionTurn ()

Destructor.

• virtual **ArActionDesired** * **fire** (**ArActionDesired** currentDesired)

Fires the action, returning what the action wants to do.

 $\bullet \ \, {\rm virtual} \ \, \mathbf{ArActionDesired} \, * \, \mathbf{getDesired} \, \, ({\rm void}) \\$

Gets what this action wants to do (for display purposes).

4.23.1 Detailed Description

Action to turn when the behaviors with more priority have limited the speed.

This action is basically made so that you can just have a ton of limiters of different kinds and types to keep speed under control, then throw this into the mix to have the robot wander. Note that the turn amount ramps up to turnAmount starting at 0 at speedStartTurn and hitting the full amount at speedFullTurn.

4.23.2 Member Function Documentation

4.23.2.1 ArActionDesired * ArActionTurn::fire (ArActionDesired currentDesired) [virtual]

Fires the action, returning what the action wants to do.

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Reimplemented from **ArAction** (p. 42).

- ArActionTurn.h
- ArActionTurn.cpp

4.24 ArACTS_1_2 Class Reference

Driver for ACTS.

#include <ArACTS.h>

Public Types

• enum ActsConstants { NUM_CHANNELS = 32, MAX_BLOBS = 10, BLOB_DATA_SIZE = 16, DATA_HEADER = NUM_CHANNELS * 4, MAX_DATA = 5300 }

Public Methods

• ArACTS_1_2 ()

Constructor.

• virtual ~**ArACTS_1_2** ()

Destructor.

• bool **openPort** (**ArRobot** *robot, const char *host="localhost", int port=5001)

Opens the connection to ACTS.

• bool **closePort** (void)

Closes the connection.

• bool **isConnected** (void)

Finds out whether there is connection.

• ArRobot * getRobot (void)

Gets the robot this class is connected to.

• void **setRobot** (**ArRobot** *robot)

Sets the robot this class is connected to.

• bool requestPacket (void)

Requests another packet.

• bool requestQuit (void)

Requests that ACTS quits.

• bool receiveBlobInfo (void)

Gets the blob information from the connection to acts.

• int getNumBlobs (int channel)

Gets the number of blobs for the given chanel.

- bool **getBlob** (int channel, int blobNumber, **ArACTSBlob** *blob)

 Gets the given blob from the given channel.
- void actsHandler (void)

A function that reads information from acts and requests packets.

• void **invert** (int width=160, int height=120)

This will make the image stats inverted (for use with an inverted camera).

Protected Methods

• int getData (char *rawData)

an iternal function to strip out the information from some bytes.

4.24.1 Detailed Description

Driver for ACTS.

4.24.2 Member Enumeration Documentation

4.24.2.1 enum ArACTS_1_2::ActsConstants

Enumeration values:

NUM_CHANNELS Number of channels there are.

MAX_BLOBS Number of blobs per channel.

BLOB_DATA_SIZE Size of the blob data.

DATA_HEADER Size of the data header.

MAX_DATA Maximum amount of data.

4.24.3 Member Function Documentation

4.24.3.1 bool ArACTS_1_2::closePort (void)

Closes the connection.

Closes the port to the ACTS server

Returns:

true if the connection was closed properly, false otherwise

4.24.3.2 bool ArACTS_1_2::getBlob (int *channel*, int *blobNumber*, ArACTSBlob * *blob*)

Gets the given blob from the given channel.

Gets the blobNumber from the channel given, fills the information for that blob into the given blob structure.

Parameters:

channel the channel to get the blob fromblobNumber the number of the blob to get from the given channelblob the blob instance to fill in with the data about the requested blob

Returns:

true if the blob instance could be filled in from the

4.24.3.3 int ArACTS_1_2::getNumBlobs (int channel)

Gets the number of blobs for the given chanel.

Returns:

the number of blobs on the channel, or -1 if the channel is invalid

4.24.3.4 void ArACTS_1_2::invert (int width = 160, int height = 120)

This will make the image stats inverted (for use with an inverted camera).

This inverts the image, but since ACTS doesn't tell this driver the height or width, you need to provide both of those for the image, default is 160x120.

Parameters:

width the width of the images acts is grabbing (pixels) height the height of the images acts is grabbing (pixels)

4.24.3.5 bool ArACTS_1_2::openPort (ArRobot * robot, const char * host = "localhost", int port = 5001)

Opens the connection to ACTS.

Opens the port to the ACTS server

Parameters:

robot the robot to attach this to, which puts a sensorInterp on the robot so that ArACTS will always have fresh data from ACTS... giving a NULL value is perfectly acceptable, in this case ArACTS will not do any processing or requesting and you'll have to use receiveBlobInfo and requestPacket (or just call actsHandler)

port the port the ACTS server is running on, default of 5001

host the host the ACTS server is running on, default is localhost (ie this machine)

Returns:

true if the connection was established, false otherwise

4.24.3.6 bool ArACTS_1_2::receiveBlobInfo (void)

Gets the blob information from the connection to acts.

Checks the connection to the ACTS server for data, if data is there it fills in the blob information, otherwise just returns false

Returns:

true if there was new data and the data could be read successfully

4.24.3.7 bool ArACTS_1_2::requestPacket (void)

Requests another packet.

Requests a packet from the ACTS server, specifically it sends the request to the acts server over its connection

Returns:

true if the command was sent succesfully, false otherwise

4.24.3.8 bool ArACTS_1_2::requestQuit (void)

Requests that ACTS quits.

Sends a command to the ACTS server requesting that ACTS quit

Returns:

true if the request was sent succesfully, false otherwise

- \bullet ArACTS.h
- ArACTS.cpp

4.25 ArACTSBlob Class Reference

A class for the acts blob.

#include <ArACTS.h>

Public Methods

• ArACTSBlob ()

Constructor.

• virtual ~ArACTSBlob ()

Destructor.

• int **getArea** (void)

Gets the number of pixels (area) covered by the blob.

• int **getXCG** (void)

Gets the X Center of Gravity of the blob.

• int getYCG (void)

Gets the Y Center of Gravity of the blob.

• int getLeft (void)

Gets the left border of the blob.

 \bullet int $\mathbf{getRight}$ (void)

Gets the right border of the blob.

• int **getTop** (void)

Gets the top border of the blob.

• int **getBottom** (void)

Gets the bottom border of the blob.

• void **setArea** (int area)

Sets the number of pixels (area) covered by the blob.

• void **setXCG** (int xcg)

Sets the X Center of Gravity of the blob.

• void **setYCG** (int ycg)

Sets the Y Center of Gravity of the blob.

- void **setLeft** (int left)

 Sets the left border of the blob.
- void **setRight** (int right)

 Sets the right border fo the blob.
- void **setTop** (int top)

 Sets the top border of the blob.
- void **setBottom** (int bottom)

 Sets the bottom border of the blob.
- void log (void)

 Prints the stats of the blob.

4.25.1 Detailed Description

A class for the acts blob.

The documentation for this class was generated from the following file:

• ArACTS.h

4.26 ArAMPTU Class Reference

Driver for the AMPUT.

 $\verb|#include| < \verb|ArAMPTU.h>|$

Inheritance diagram for ArAMPTU::



Public Methods

- **ArAMPTU** (**ArRobot** *robot, int unitNumber=0) Constructor.
- virtual ~**ArAMPTU** ()

 Destructor.
- virtual bool **init** (void)

 Initializes the camera.
- virtual bool **pan** (int deg)

 Pans to the given degrees.
- virtual bool panRel (int deg)
 Pans relative to current position by given degrees.
- virtual bool **tilt** (int deg)

 Tilts to the given degrees.
- virtual bool **tiltRel** (int deg)

 Tilts relative to the current position by given degrees.
- virtual bool **panTilt** (int panDeg, int tiltDeg)

 Pans and tilts to the given degrees.
- virtual bool **panTiltRel** (int panDeg, int tiltDeg)

 Pans and tilts relatives to the current position by the given degrees.

• bool **panSlew** (int deg)

Sets the rate that the camera pans at.

• bool tiltSlew (int deg)

Sets the rate the camera tilts at.

• virtual bool canZoom (void) const

Returns true if camera can zoom (or rather, if it is controlled by this).

• virtual int getMaxPosPan (void) const

Gets the highest positive degree the camera can pan to.

• virtual int **getMaxNegPan** (void) const

Gets the lowest negative degree the camera can pan to.

ullet virtual int $\mathbf{getMaxPosTilt}$ (void) const

Gets the highest positive degree the camera can tilt to.

• virtual int **getMaxNegTilt** (void) const

Gets the lowest negative degree the camera can tilt to.

• bool **pause** (void)

Stops current pan/tilt, can be resumed later.

• bool **resume** (void)

Resumes a previously paused pan/tilt.

• bool **purge** (void)

Stops motion and purges last command.

• bool requestStatus (void)

Retrieves the camera status.

• virtual int getPan (void) const

Gets the angle the camera is panned to.

• virtual int **getTilt** (void) const

Gets the angle the camera is tilted to.

4.26.1 Detailed Description

Driver for the AMPUT.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 ArAMPTU::ArAMPTU (ArRobot * robot, int unitNumber = 0)

Constructor.

Parameters:

robot the robot to attach to unitNumber the unit number for this packet, this needs to be 0-7

- \bullet ArAMPTU.h
- $\bullet \ \, {\rm ArAMPTU.cpp}$

4.27 ArAMPTUCommands Class Reference

A class with the commands for the AMPTU.

#include <ArAMPTU.h>

Public Types

• enum { ABSTILT = 0x35, RELTILTU = 0x36, RELTILTD = 0x37, ABSPAN = 0x31, RELPANCW = 0x32, RELPANCCW = 0x33, PANTILT = 0x28, PANTILTUCW = 0x29, PANTILTDCW = 0x2A, PANTILTUCCW = 0x2B, PANTILTDCCW = 0x2C, ZOOM = 0x3F, PAUSE = 0x39, CONT = 0x3A, PURGE = 0x3B, STATUS = 0x3C, INIT = 0x3D, RESP = 0x3E, PANSLEW = 0x34, TILT-SLEW = 0x38 }

4.27.1 Detailed Description

A class with the commands for the AMPTU.

4.27.2 Member Enumeration Documentation

4.27.2.1 anonymous enum

Enumeration values:

ABSTILT Absolute tilt.

RELTILTU Relative tilt, up.

RELTILTD Relative tilt, down.

ABSPAN Absolute pan.

RELPANCW Relative pan, clockwise.

RELPANCCW Relative pan, counter clockwise.

PANTILT Pan and tilt absolute.

PANTILTUCW Relative tilt up, pan clockwise.

PANTILTDCW Relative tilt down, pan clockwise.

PANTILTUCCW Relative tilt up, pan counter-clockwise.

PANTILTDCCW Relative tilt down, pan counter-clockwise.

ZOOM Zoom.

PAUSE Pause the current movement.

CONT Continue paused movement.

PURGE Stops movement and purges commands.

STATUS Requests a status packet.

INIT Initializes the camera.

RESP Response.

PANSLEW Sets the pan slew rate.

TILTSLEW Sets the tilt slew rate.

The documentation for this class was generated from the following file:

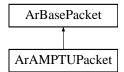
 \bullet ArAMPTU.h

4.28 ArAMPTUPacket Class Reference

A class for for making commands to send to the AMPTU.

#include <ArAMPTU.h>

Inheritance diagram for ArAMPTUPacket::



Public Methods

 $\bullet \ \mathbf{ArAMPTUPacket} \ (\mathbf{ArTypes::} \mathbf{UByte2} \ \mathbf{bufferSize} {=} 30)$

Constructor.

• virtual ~ArAMPTUPacket ()

Destructor.

• unsigned char **getUnitNumber** (void)

Gets the unit number this packet is for.

• bool **setUnitNumber** (unsigned char unitNumber)

Sets the unit number htis packet is for.

• virtual void byteToBuf (ArTypes::Byte val)

Puts ArTypes::Byte (p. 496) into packets buffer.

• virtual void byte2ToBuf (ArTypes::Byte2 val)

Puts ArTypes::Byte2 (p. 496) into packets buffer.

• virtual void **finalizePacket** (void)

MakeFinals the packet in preparation for sending, must be done.

4.28.1 Detailed Description

A class for for making commands to send to the AMPTU.

There are only a few functioning ways to put things into this packet, you MUST use thse, if you use anything else your commands won't work. You must use byteToBuf and byte2ToBuf.

See also:

getUnitNumber (p. 102), setUnitNumber (p. 102)

4.28.2 Member Function Documentation

4.28.2.1 unsigned char ArAMPTUPacket::getUnitNumber (void)

Gets the unit number this packet is for.

Each AMPTU has a unit number, so that you can daisy chain multiple ones together. This number is incorporated into the packet header, thus the packet has to know what the number is.

Returns:

the unit number this packet has

4.28.2.2 bool ArAMPTUPacket::setUnitNumber (unsigned char unitNumber)

Sets the unit number htis packet is for.

Each AMPTU has a unit number, so that you can daisy chain multiple ones together. This number is incorporated into the packet header, thus the packet has to know what the number is.

Parameters:

unitNumber the unit number for this packet, this needs to be 0-7

Returns

true if the number is acceptable, false otherwise

- ArAMPTU.h
- ArAMPTU.cpp

4.29 ArArg Class Reference

Argument class, mostly for actions, could be used for other things.

```
#include <ArArg.h>
```

Public Types

Public Methods

- **ArArg** ()
 - Default empty contructor.
- **ArArg** (const char *name, int *pointer, const char *description="")

 Constructor for making an integer argument.
- **ArArg** (const char *name, double *pointer, const char *description="") Constructor for making a double argument.
- **ArArg** (const char *name, std::string *pointer, const char *description="")

Constructor for making a string argument.

- **ArArg** (const char *name, bool *pointer, const char *description="")

 Constructor for making a boolean argument.
- ArArg (const char *name, ArPose *pointer, const char *description="")

 Constructor for making a position argument.

Copy constructor.

• virtual ~ArArg ()

Destructor.

- Type getType (void) const Gets the type of the argument.
- const char * **getName** (void) const

Gets the name of the argument.

- const char * **getDescription** (void) const Gets the long description of the argument.
- void **setInt** (int val)

 Sets the argument value, for int arguments.
- void setDouble (double val)
 Sets the argument value, for double arguments.
- void **setString** (const char *str)

 Sets the argument value, for string arguments.
- void setBool (bool val)
 Sets the argument value, for bool arguments.
- void **setPose** (**ArPose** pose)

 Sets the argument value, for **ArPose** (p. 285) arguments.
- int **getInt** (void) const

 Gets the argument value, for int arguments.
- double **getDouble** (void) const

 Gets the argument value, for double arguments.
- const char * **getString** (void) const

 Gets the argument value, for string arguments.
- bool **getBool** (void) const

 Gets the argument value, for bool arguments.
- ArPose getPose (void) const Gets the argument value, for pose arguments.
- void log (void) const
 Logs the type, name, and value of this argument.
- void **clearPointers** (void)

 Internal helper function.

4.29.1 Detailed Description

Argument class, mostly for actions, could be used for other things.

This is designed to be easy to add another type to the arguments... All you have to do to do so, is add an enum to the Type enum, add a newType getNewType(void), add a void setNewType(newType nt), and add a case statement for the newType to ArArg::print. You should probably also add an

See also:

newType to the documentation for ArArg::getType (p. 105).

4.29.2 Member Enumeration Documentation

4.29.2.1 enum ArArg::Type

Enumeration values:

INVALID An invalid argument, the argument wasn't created correctly.

INT Integer argument.

DOUBLE Double argument.

STRING String argument.

BOOL Boolean argument.

POSE ArPose (p. 285) argument.

4.29.3 Member Function Documentation

4.29.3.1 ArArg::Type ArArg::getType (void) const

Gets the type of the argument.

See also:

```
INVALID (p. 105), INT (p. 105), DOUBLE (p. 105), STRING (p. 105), BOOL (p. 105), POSE (p. 105)
```

- ArArg.h
- ArArg.cpp

4.30 ArArgumentBuilder Class Reference

This class is to build arguments for things that require argc and argv. #include <ariaUtil.h>

Public Methods

 $\bullet \ \mathbf{ArArgumentBuilder} \ (\mathbf{size_t} \ \mathbf{argvLen} {=} 256)$

Constructor.

• virtual ~**ArArgumentBuilder** ()

Destructor.

• void add (char *str,...)

Adds the given string, with varargs, seperates if there are spaces.

• void addPlain (char *str)

Adds the given string, without varargs (wrapper for java).

• void **log** (void) const

Prints out the arguments.

• size_t **getArgc** (void) const Gets the argc.

• char ** **getArgv** (void) const Gets the argv.

void removeArg (size_t which)
 Delete a particular arg, you MUST finish adding before you can remove.

4.30.1 Detailed Description

This class is to build arguments for things that require argc and argv.

- ariaUtil.h
- ariaUtil.cpp

4.31 ArArgumentParser Class Reference

Class for parsing arguments.

#include <ariaUtil.h>

Public Methods

• ArArgumentParser (int *argc, char **argv)

Constructor, takes the argc argv.

• ArArgumentParser (ArArgumentBuilder *builder)

Constructor, takes an argument builder.

• ~ArArgumentParser ()

Destructor.

• bool **checkArgument** (char *argument)

Returns true if the argument was found.

• char * checkParameterArgument (char *argument)

Returns the word/argument after given argument or NULL if it is not present.

• size_t **getArgc** (void) const

Gets how many arguments are left in this parser.

• void log (void) const

Prints out the arguments left in this parser.

4.31.1 Detailed Description

Class for parsing arguments.

This class is made for parsing arguments form an argc/argv set... if you're using a winmain you can first toss your string at the **ArArgumentBuilder** (p. 106) above class ArArgumentParser and then use this parser on it

4.31.2 Constructor & Destructor Documentation

4.31.2.1 ArArgumentParser::ArArgumentParser (int * argc, char ** argv)

Constructor, takes the argc argv.

Parameters:

argc a pointer to the argc used argv argv

4.31.2.2 ArArgumentParser::ArArgumentParser (ArArgumentBuilder * builder)

Constructor, takes an argument builder.

Parameters:

argc a pointer to the argc used argv argv

4.31.3 Member Function Documentation

4.31.3.1 bool ArArgumentParser::checkArgument (char * argument)

Returns true if the argument was found.

Parameters:

argument the string to check for, if the argument is found its pulled from the list of arguments

Returns:

true if the argument was found, false otherwise

4.31.3.2 char * ArArgumentParser::checkParameterArgument (char * argument)

Returns the word/argument after given argument or NULL if it is not present.

Parameters:

argument the string to check for, if the argument is found its pulled from the list of arguments

Returns:

NULL if the argument wasn't found, the argument after the one given if the argument was found, or NULL again if the argument was found as the last item

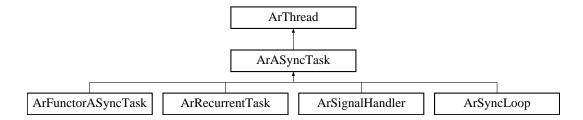
- \bullet ariaUtil.h
- $\bullet \ \, {\rm ariaUtil.cpp}$

4.32 ArASyncTask Class Reference

Asynchronous task (runs in its own thread).

#include <ArASyncTask.h>

Inheritance diagram for ArASyncTask::



Public Methods

• ArASyncTask ()

Constructor.

• virtual ~ArASyncTask ()

Destructor.

- virtual void * runThread (void *arg)=0

 The main run loop.
- virtual void **run** (void)

Run in this thread.

• virtual void **runAsync** (void)

Run in its own thread.

 $\bullet \ {\rm virtual \ void \ stopRunning \ (void)}$

Stop the thread.

• virtual int **create** (bool joinable=true, bool lowerPriority=true)

Create the task and start it going.

• virtual void * runInThisThread (void *arg=0)

Run the code of the task syncronously.

4.32.1 Detailed Description

Asynchronous task (runs in its own thread).

The ArAsynTask is a task that runs in its own thread. This is a rather simple class. The user simply needs to derive their own class from ArAsyncTask and define the **runThread()** (p. 111) function. They then need to create an instance of their task and call run or runAsync. The standard way to stop a task is to call **stopRunning()** (p. 110) which sets **ArThread::myRunning(** p. 489) to false. In their run loop, they should pay attention to the **getRunning()** (p. 488) or the **ArThread::myRunning()** (p. 489) variable. If this value goes to false, the task should clean up after itself and exit its **runThread()** (p. 111) function.

4.32.2 Member Function Documentation

4.32.2.1 void * ArASyncTask::runInThisThread (void * arg = 0) [virtual]

Run the code of the task syncronously.

This will run the code of the ArASyncTask without creating a new thread to run it in. It performs the needed setup then calls **runThread**() (p. 111). This is good if you have a task which you wish to run multiple instances of and you want to use the main() thread instead of having it block, waiting for exit of the program.

Parameters:

arg the argument to pass to the runThread() (p. 111)

4.32.2.2 virtual void* ArASyncTask::runThread (void * arg) [pure virtual]

The main run loop.

Override this function and put your taskes run loop here. Check the value of **getRunning**() (p. 488) or myRunning periodicly in your loop. If the value goes false, the loop should exit and **runThread**() (p. 111) should return.

Reimplemented in **ArFunctorASyncTask** (p. 164), **ArRecurrentTask** (p. 324), and **ArSignalHandler** (p. 457).

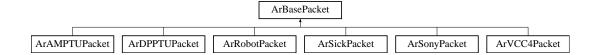
- ArASyncTask.h
- ArASyncTask.cpp

4.33 ArBasePacket Class Reference

Base packet class.

#include <ArBasePacket.h>

Inheritance diagram for ArBasePacket::



Public Methods

• ArBasePacket (ArTypes::UByte2 bufferSize=0, ArTypes::UByte2 headerLength=0, char *buf=NULL, ArTypes::UByte2 footer-Length=0)

Constructor.

- virtual ~ArBasePacket ()
 - Destructor.
- virtual void **empty** (void)

 resets the length for more data to be added.
- virtual void finalizePacket (void)
 MakeFinals the packet in preparation for sending, must be done.
- virtual void log (void)

 ArLogs the contents of the packet.
- $\bullet \ \, {\rm virtual} \,\, {\rm void} \,\, {\bf printHex} \,\, ({\rm void})$

ArLogs the contents of the packet in hex.

- virtual void byteToBuf (ArTypes::Byte val)

 Puts ArTypes::Byte (p. 496) into packets buffer.
- virtual void byte2ToBuf (ArTypes::Byte2 val)

 Puts ArTypes::Byte2 (p. 496) into packets buffer.
- virtual void byte4ToBuf (ArTypes::Byte4 val)

 Puts ArTypes::Byte4 (p. 496) into packets buffer.

- virtual void uByteToBuf (ArTypes::UByte val)
 Puts ArTypes::UByte (p. 496) into packets buffer.
- virtual void uByte2ToBuf (ArTypes::UByte2 val)
 Puts ArTypes::UByte2 (p. 496) into packet buffer.
- virtual void uByte4ToBuf (ArTypes::UByte4 val)

 Puts ArTypes::UByte (p. 496) 4 into packet buffer.
- virtual void **strToBuf** (const char *str)

 Puts a string into packet buffer.
- virtual void **strNToBuf** (const char *str, int length)

 Copies length bytes from str into packet buffer.
- virtual void **strToBufPadded** (const char *str, int length)

 Copies length bytes from str, if str ends before length, pads data.
- virtual void **dataToBuf** (const char *data, int length)

 Copies length bytes from data into packet buffer.
- virtual ArTypes::Byte bufToByte (void)
 Gets a ArTypes::Byte (p. 496) from the buffer.
- virtual ArTypes::Byte2 bufToByte2 (void)

 Gets a ArTypes::Byte2 (p. 496) from the buffer.
- virtual ArTypes::Byte4 bufToByte4 (void)
 Gets a ArTypes::Byte4 (p. 496) from the buffer.
- virtual **ArTypes::UByte bufToUByte** (void)

 Gets a **ArTypes::UByte** (p. 496) from the buffer.
- virtual ArTypes::UByte2 bufToUByte2 (void)

 Gets a ArTypes::UByte2 (p. 496) from the buffer.
- virtual ArTypes::UByte4 bufToUByte4 (void)

 Gets a ArTypes::UByte4 (p. 496) from the buffer.
- virtual void **bufToStr** (char *buf, int len)

 Gets a string from the buffer.

- virtual void bufToData (char *data, int length)
 Gets length bytes from buffer and puts them into data.
- virtual void **resetRead** (void)

 Restart the reading process.
- virtual ArTypes::UByte2 getLength (void)

 Gets the total length of the packet.
- virtual ArTypes::UByte2 getDataLength (void)

 Gets the length of the data in the packet.
- virtual ArTypes::UByte2 getReadLength (void)

 Gets how far into the packet that has been read.
- virtual ArTypes::UByte2 getDataReadLength (void)

 Gets how far into the data of the packet that has been read.
- virtual ArTypes::UByte2 getHeaderLength (void)

 Gets the length of the header.
- virtual **ArTypes::UByte2 getFooterLength** (void) Gets the length of the header.
- virtual ArTypes::UByte2 getMaxLength (void)

 Gets the maximum length packet.
- virtual const char * **getBuf** (void)

 Gets a pointer to the buffer the packet uses.
- virtual void **setBuf** (char *buf)

 Sets the buffer the packet is using.
- virtual bool **setLength** (**ArTypes::UByte2** length)

 Sets the length of the packet.
- virtual void **setReadLength** (**ArTypes::UByte2** readLength)

 Sets the read length.
- virtual bool **setHeaderLength** (**ArTypes::UByte2** length)

 Sets the length of the header.

• virtual void **duplicatePacket** (ArBasePacket *packet)

Makes this packet a duplicate of another packet.

4.33.1 Detailed Description

Base packet class.

This class is a base class for all packets... most software will never need to use this class, it is there mostly to help people do more advanced client and server communications.

All of the functions are virtual so it can be completely overridden if desired... but the few most likely ones to be overridden are empty and makeFinal...

The theory of the packet works like this, the packet has a buffer, headerLength, readLength, length, and a maxLength. When the packet is initialized it is given a buffer and its maxLength. All of the functions that are somethingToBuf put data in at the current length of the packet, and advance the length. All of the functions that do bufToSomething get the data from where readLength points, and advance read length. resetRead sets readLength back to the header (since no one outside of the person who writes the class should touch the header). empty likewise sets the length back to the header since the header will be calculated in the finalizePacket method.

The base class and most classes of this kind will have an integer before the string, denoting the strings length... this is hidden by the function calls, but something someone may want to be aware of... it should not matter much as this same packet class should be used on both sides.

Uses of this class that don't get newed and deleted a lot can just go ahead and use the constructor with buf = NULL, as this will have the packet manage its own memory, making life easier.

4.33.2 Constructor & Destructor Documentation

```
4.33.2.1 ArBasePacket::ArBasePacket (ArTypes::UByte2 bufferSize = 0, ArTypes::UByte2 headerLength = 0, char * buf = NULL, ArTypes::UByte2 footerLength = 0)
```

Constructor.

Parameters:

bufferSize size of the buffer

headerLength length of the header

buf buffer packet uses, if NULL, instance will allocate memory

4.33.3 Member Function Documentation

4.33.3.1 void ArBasePacket::bufToData (char * data, int length) [virtual]

Gets length bytes from buffer and puts them into data.

copies length bytes from the buffer into data, length is passed in, not read from packet

Parameters:

data character array to copy the data into length number of bytes to copy into data

4.33.3.2 void ArBasePacket::bufToStr (char * buf, int len) [virtual]

Gets a string from the buffer.

puts a string from the packets buffer into the given buffer, stopping when it reaches the end of the packets buffer or the length of the given buffer or a '\0'

4.33.3.3 void ArBasePacket::dataToBuf (const char * data, int length) [virtual]

Copies length bytes from data into packet buffer.

puts data into the buffer without putting in length first

Parameters:

data chacter array to copy into buffer

legnth how many botes to copy from data into packet

4.33.3.4 void ArBasePacket::duplicatePacket (ArBasePacket * packet) [virtual]

Makes this packet a duplicate of another packet.

Copies the given packets buffer into the buffer of this packet, also sets this length and readlength to what the given packet has

Parameters:

packet the packet to duplicate

4.33.3.5 void ArBasePacket::empty (void) [virtual]

resets the length for more data to be added.

Sets the packet length back to be the packets header length again

4.33.3.6 void ArBasePacket::resetRead (void) [virtual]

Restart the reading process.

Sets the length read back to the header length so the packet can be reread using the other methods

Reimplemented in **ArSickPacket** (p. 449).

4.33.3.7 void ArBasePacket::strNToBuf (const char * str, int length) [virtual]

Copies length bytes from str into packet buffer.

first puts the length of the string into the buffer, then puts in string

Parameters:

str character array to copy into buffer

<code>length</code> how many bytes to copy from the str into packet

4.33.3.8 void ArBasePacket::strToBuf (const char * str) [virtual]

Puts a string into packet buffer.

first puts the length of the string into the buffer, then puts in string

Parameters:

str string to copy into buffer

4.33.3.9 void ArBasePacket::strToBufPadded (const char * str, int length) [virtual]

Copies length bytes from str, if str ends before length, pads data.

first puts the length of the string into the buffer, then puts in string, if string ends before length it pads the string

Parameters:

str character array to copy into buffer

length how many bytes to copy from the str into packet

- ArBasePacket.h
- ArBasePacket.cpp

4.34 ArCommands Class Reference

A class with an enum of the commands that can be sent to the robot.

#include <ArCommands.h>

Public Types

enum Commands { PULSE = 0, OPEN = 1, CLOSE = 2, POLLING = 3, ENABLE = 4, SETA = 5, SETV = 6, SETO = 7, MOVE = 8, ROTATE = 9, SETRV = 10, VEL = 11, HEAD = 12, DHEAD = 13, SAY = 15, CONFIG = 18, ENCODER = 19, RVEL = 21, DCHEAD = 22, SETRA = 23, SONAR = 28, STOP = 29, DIGOUT = 30, VEL2 = 32, GRIPPER = 33, ADSEL = 35, GRIPPERVAL = 36, GRIPPERPACREQUEST = 37, IOREQUEST = 40, PTUPOS = 41, TTY2 = 42, GETAUX = 43, BUMPSTALL = 44, TCM2 = 45, JOYDRIVE = 47, ESTOP = 55, LOADPARAM = 61, ENDSIM = 62, LOADWORLD = 63, STEP = 64, CALCOMP = 65, SETSIMORIGINX = 66, SETSIMORIGINY = 67, SETSIMORIGINTH = 68, RESETSIMTOORIGIN = 69, SOUND = 90, PLAYLIST = 91, SOUNDTOG = 92 }

4.34.1 Detailed Description

A class with an enum of the commands that can be sent to the robot.

A class with an enum of the commands that can be sent to the robot, see the operations manual for more detailed descriptions.

4.34.2 Member Enumeration Documentation

4.34.2.1 enum ArCommands::Commands

Enumeration values:

PULSE none, keep alive command, so watchdog doesn't trigger.

OPEN none, sent after connection to initiate connection.

CLOSE none, sent to close the connection to the robot.

POLLING string, string that sets sonar polling sequence.

ENABLE int, enable (1) or disable (0) motors.

SETA int, sets translational accel (+) or decel (-) (mm/sec/sec).

SETV int, sets maximum velocity (mm/sec).

SETO int, resets robots origin back to 0, 0, 0.

MOVE int, translational move (mm).

ROTATE int, set rotational velocity, duplicate of RVEL (deg/sec).

SETRV int, sets the maximum rotational velocity (deg/sec).

VEL int, set the translational velocity (mm/sec).

HEAD int, turn to absolute heading 0-359 (degrees).

DHEAD int, turn relative to current heading (degrees).

SAY string, makes the robot beep. up to 20 pairs of duration (20 ms incrs) and tones (halfcycle)

CONFIG int, request configuration packet.

ENCODER int, > 0 to request continuous stream of packets, 0 to stop.

RVEL int, set rotational velocity (deg/sec).

DCHEAD int, colbert relative heading setpoint (degrees).

SETRA int, sets rotational accel(+) or decel(-) (deg/sec).

SONAR int, enable (1) or disable (0) sonar.

STOP int, stops the robot.

DIGOUT int, sets the digout lines.

VEL2 2bytes, independent wheel velocities, first byte = right, second = left

GRIPPER int, gripper server command, see gripper manual for detail.

ADSEL int, select the port given as argument.

GRIPPERVAL p2 gripper server value, see gripper manual for details.

GRIPPERPACREQUEST p2 gripper packet request.

IOREQUEST request iopackets from p2os.

PTUPOS most-sig byte is port number, least-sig byte is pulse width.

TTY2 string, send string argument to serial dev connected to aux1.

GETAUX int, requests 1-200 bytes from aux1 serial channel, 0 flush.

BUMPSTALL int, stop and register a stall if front (1), rear (2), or both (3) bump rings are triggered, Off (default) is 0

TCM2 TCM2 module commands, see p2 tcm2 manual for details.

JOYDRIVE Command to tell p2os to drive with the joystick plugged into the robot

ESTOP none, emergency stop, overrides decel.

LOADPARAM string, Sim Specific, causes the sim to load the given param file.

ENDSIM none, Sim Specific, causes the simulator to close and exit.

LOADWORLD string, Sim Specific, causes the sim to load given world.

STEP none, Sim Specific, single step mode.

CALCOMP int, commands for calibrating compass, see compass manual.

SETSIMORIGINX int, sets the X origin in the simulator.

SETSIMORIGINY int, sets the Y origin in the simulator.

SETSIMORIGINTH int, sets the heading at origin in the simulator.

RESETSIMTOORIGIN int, resets the sim robots poseiton to origin.

SOUND int, AmigoBot specific, plays sound with given number.

PLAYLIST int, AmigoBot specific, requests name of sound, 0 for all, otherwise for specific sound

SOUNDTOG int, AmigoBot specific, enable(1) or diable(0) sound.

The documentation for this class was generated from the following file:

• ArCommands.h

4.35 ArCondition Class Reference

Threading condition wrapper class.

#include <ArCondition.h>

Public Types

enum typedef { STATUS_FAILED = 1, STATUS_FAILED_DESTROY, STATUS_FAILED_INIT, STATUS_WAIT_TIMEDOUT, STATUS_WAIT_INTR, STATUS_MUTEX_FAILED_INIT, STATUS_MUTEX_FAILED }

Public Methods

• ArCondition ()

Constructor.

- virtual ~**ArCondition** ()
 - Desctructor.
- int signal ()

Signal the thread waiting.

• int broadcast ()

Broadcast a signal to all threads waiting.

• int wait ()

Wait for a signal.

• int timedWait (unsigned int msecs)

Wait for a signal for a period of time in milliseconds.

• const char * **getError** (int messageNumber) const

Translate error into string.

4.35.1 Detailed Description

Threading condition wrapper class.

4.35.2 Member Enumeration Documentation

4.35.2.1 enum ArCondition::typedef

Enumeration values:

STATUS_FAILED General failure.

STATUS_FAILED_DESTROY Another thread is waiting on this condition so it can not be destroyed.

STATUS_FAILED_INIT Failed to initialize thread. Requested action is imposesible.

STATUS_WAIT_TIMEDOUT The timedwait timed out before signaling.

STATUS_WAIT_INTR The wait was interupted by a signal.

STATUS_MUTEX_FAILED_INIT The underlying mutex failed to init.

STATUS_MUTEX_FAILED The underlying mutex failed in some fashion.

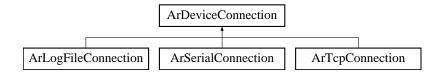
- ArCondition.h
- ArCondition_LIN.cpp
- $\bullet \ \, ArCondition_WIN.cpp$

4.36 ArDeviceConnection Class Reference

Base class for device connections.

#include <ArDeviceConnection.h>

Inheritance diagram for ArDeviceConnection::



Public Types

 enum Status { STATUS_NEVER_OPENED = 1, STATUS_OPEN, STATUS_OPEN_FAILED, STATUS_CLOSED_NORMALLY, STATUS_CLOSED_ERROR }

Public Methods

- $\bullet \ \mathbf{ArDeviceConnection} \ () \\$
 - constructor.
- virtual ~ArDeviceConnection ()
 destructor also forces a close on the connection.
- virtual int **read** (const char *data, unsigned int size, unsigned int ms-Wait=0)=0

 $Reads\ data\ from\ connection.$

- virtual int writePacket (ArBasePacket *packet)
 - Writes data to connection.
- virtual int write (const char *data, unsigned int size)=0

 Writes data to connection.
- virtual int **getStatus** (void)=0

 Gets the status of the connection, which is one of the enum status.
- const char * **getStatusMessage** (int messageNumber) const Gets the description string associated with the status.

• virtual bool **openSimple** (void)=0

Opens the connection again, using the values from setLocation or.

• virtual bool **close** (void)

Closes the connection.

• virtual const char * **getOpenMessage** (int messageNumber)=0

Gets the string of the message associated with opening the device.

• virtual **ArTime getTimeRead** (int index)=0

Gets the time data was read in.

• virtual bool **isTimeStamping** (void)=0

sees if timestamping is really going on or not.

4.36.1 Detailed Description

Base class for device connections.

Base class for device connections, this is mostly for connections to the robot or simulator but could also be used for a connection to a laser or other device

Note that this is mostly a base class, so if you'll want to use one of the classes which inherit from this one... also note that in those classes is where you'll find setPort which sets the place the device connection will try to connect to... the inherited classes also have an open which returns more detailed information about the open attempt, and which takes the parameters for where to connect

4.36.2 Member Enumeration Documentation

4.36.2.1 enum ArDeviceConnection::Status

Enumeration values:

STATUS_NEVER_OPENED Never opened.

STATUS_OPEN Currently open.

STATUS_OPEN_FAILED Tried to open, but failed.

STATUS_CLOSED_NORMALLY Closed by a close call.

STATUS_CLOSED_ERROR Closed because of error.

4.36.3 Member Function Documentation

4.36.3.1 virtual bool ArDeviceConnection::close (void) [inline, virtual]

Closes the connection.

Returns:

whether the close succeeded or not

Reimplemented in ArLogFileConnection (p. 234), ArSerialConnection (p. 426), and ArTcpConnection (p. 483).

4.36.3.2 virtual const char* ArDeviceConnection::getOpenMessage (int messageNumber) [pure virtual]

Gets the string of the message associated with opening the device.

Each class inherited from this one has an open method which returns 0 for success or an integer which can be passed into this function to obtain a string describing the reason for failure

Parameters:

messageNumber the number returned from the open

Returns:

the error description associated with the messageNumber

Reimplemented in ArLogFileConnection (p. 235), ArSerialConnection (p. 427), and ArTcpConnection (p. 483).

4.36.3.3 virtual int ArDeviceConnection::getStatus (void) [pure virtual]

Gets the status of the connection, which is one of the enum status.

Gets the status of the connection, which is one of the enum status. If you want to get a string to go along with the number, use getStatusMessage

Returns

the status of the connection

See also:

getStatusMessage (p. 127)

Reimplemented in ArLogFileConnection (p. 235), ArSerialConnection (p. 427), and ArTcpConnection (p. 484).

4.36.3.4 const char * ArDeviceConnection::getStatusMessage (int messageNumber) const

Gets the description string associated with the status.

Parameters:

messageNumber the int from getStatus you want the string for

Returns:

the description associated with the status

See also:

getStatus (p. 126)

4.36.3.5 virtual ArTime ArDeviceConnection::getTimeRead (int index) [pure virtual]

Gets the time data was read in.

Parameters:

index looks like this is the index back in the number of bytes last read in

Returns:

the time the last read data was read in

Reimplemented in ArLogFileConnection (p. 235), ArSerialConnection (p. 427), and ArTcpConnection (p. 484).

4.36.3.6 virtual bool ArDeviceConnection::isTimeStamping (void) [pure virtual]

sees if timestamping is really going on or not.

Returns:

true if real timestamping is happening, false otherwise

Reimplemented in ArLogFileConnection (p. 236), ArSerialConnection (p. 428), and ArTcpConnection (p. 484).

4.36.3.7 virtual int ArDeviceConnection::read (const char * data, unsigned int size, unsigned int msWait = 0) [pure virtual]

Reads data from connection.

Reads data from connection

Parameters:

data pointer to a character array to read the data into

size maximum number of bytes to read

msWait read blocks for this many milliseconds (not at all for < 0)

Returns:

number of bytes read, or -1 for failure

See also:

write (p. 128), writePacket (p. 129)

Reimplemented in ArLogFileConnection (p. 236), ArSerialConnection (p. 428), and ArTcpConnection (p. 485).

4.36.3.8 virtual int ArDeviceConnection::write (const char * data, unsigned int size) [pure virtual]

Writes data to connection.

Writes data to connection

Parameters:

data pointer to a character array to write the data from

size number of bytes to write

Returns:

number of bytes read, or -1 for failure

See also:

 $\mathbf{read} \ (\mathrm{p.\,}128), \ \mathbf{writePacket} \ (\mathrm{p.\,}129)$

Reimplemented in ArLogFileConnection (p. 237), ArSerialConnection (p. 430), and ArTcpConnection (p. 486).

4.36.3.9 virtual int ArDeviceConnection::writePacket (ArBasePacket * packet) [inline, virtual]

Writes data to connection.

Writes data to connection from a packet

Parameters:

packet pointer to a packet to write the data from

Returns:

number of bytes written, or -1 for failure

See also:

read (p. 128), write (p. 128)

- ArDeviceConnection.h
- ArDeviceConnection.cpp

4.37 ArDPPTU Class Reference

Driver for the DPPTU.

#include <ArDPPTU.h>

Inheritance diagram for ArDPPTU::



Public Types

enum { , MIN_PAN = -158, MAX_TILT = 30, MIN_TILT = -46, MAX_PAN_SLEW = 149, MIN_PAN_SLEW = 2, MAX_TILT_SLEW = 149, MIN_TILT_SLEW = 2, MAX_PAN_ACCEL = 102, MIN_PAN_ACCEL = 2, MAX_TILT_ACCEL = 102, MIN_TILT_ACCEL = 2 }

Public Methods

• ArDPPTU (ArRobot *robot)

Constructor.

• virtual ~ArDPPTU ()

Destructor.

• bool init (void)

Initializes the camera.

ullet bool **canZoom** (void) const

Returns true if camera can zoom (or rather, if it is controlled by this).

• bool **blank** (void)

Sends a delimiter only.

• bool resetCalib (void)

Perform reset calibration.

• bool disableReset (void)

Disable power-on reset.

• bool resetTilt (void)

Reset tilt axis.

• bool **resetPan** (void)

Reset pan axis only.

• bool **resetAll** (void)

Reset pan and tilt axes on power-on.

• bool saveSet (void)

Save current settings as defaults.

• bool restoreSet (void)

Restore stored defaults.

• bool **factorySet** (void)

Restore factory defaults.

• bool panTilt (int pdeg, int tdeg)

Pans and tilts to the given degrees.

• bool pan (int deg)

Pans to the given degrees.

• bool panRel (int deg)

Pans relative to current position by given degrees.

• bool tilt (int deg)

Tilts to the given degrees.

• bool tiltRel (int deg)

Tilts relative to the current position by given degrees.

• bool panTiltRel (int pdeg, int tdeg)

Pans and tilts relatives to the current position by the given degrees.

• bool limitEnforce (bool val)

Enables or disables the position limit enforcement.

• bool immedExec (void)

Sets unit to immediate-execution mode for positional commands.

• bool **slaveExec** (void)

Sets unit to slaved-execution mode for positional commands.

• bool awaitExec (void)

Instructs unit to await completion of the last issued command.

• bool haltAll (void)

Halts all pan-tilt movement.

• bool haltPan (void)

Halts pan axis movement.

• bool haltTilt (void)

Halts tilt axis movement.

• virtual int **getMaxPosPan** (void) const

Gets the highest positive degree the camera can pan to.

• virtual int **getMaxNegPan** (void) const

Gets the lowest negative degree the camera can pan to.

• virtual int **getMaxPosTilt** (void) const

Gets the highest positive degree the camera can tilt to.

• virtual int **getMaxNegTilt** (void) const Gets the lowest negative degree the camera can tilt to.

- bool initMon (int deg1, int deg2, int deg3, int deg4)

 Sets monitor mode pan pos1/pos2, tilt pos1/pos2.
- bool **enMon** (void)

Enables monitor mode at power up.

• bool **disMon** (void)

Disables monitor mode at power up.

• bool **offStatPower** (void)

Sets stationary power mode to off.

• bool **regStatPower** (void)

Sets regular stationary power mode.

• bool lowStatPower (void)

Sets low stationary power mode.

• bool **highMotPower** (void)

Sets high in-motion power mode.

• bool regMotPower (void)

 $Sets\ regular\ in\text{-}motion\ power\ mode.$

• bool lowMotPower (void)

 $Sets\ low\ in\text{-}motion\ power\ mode.$

• bool panAccel (int deg)

Sets acceleration for pan axis.

• bool **tiltAccel** (int deg)

Sets acceleration for tilt axis.

• bool basePanSlew (int deg)

Sets the start-up pan slew.

• bool baseTiltSlew (int deg)

Sets the start-up tilt slew.

• bool **upperPanSlew** (int deg)

Sets the upper pan slew.

• bool lowerPanSlew (int deg)

Sets the lower pan slew.

• bool upperTiltSlew (int deg)

Sets the upper tilt slew.

• bool lowerTiltSlew (int deg)

Sets the lower pan slew.

• bool **indepMove** (void)

Sets motion to independent control mode.

• bool **velMove** (void)

Sets motion to pure velocity control mode.

• bool panSlew (int deg)

Sets the rate that the unit pans at.

• bool tiltSlew (int deg)

Sets the rate the unit tilts at.

• bool **panSlewRel** (int deg)

Sets the rate that the unit pans at, relative to current slew.

• bool tiltSlewRel (int deg)

Sets the rate the unit tilts at, relative to current slew.

• virtual int getPan (void) const

The angle the camera was last told to pan to.

• virtual int **getTilt** (void) const

The angle the camera was last told to tilt to.

• int **getPanSlew** (void)

Gets the current pan slew.

• int **getTiltSlew** (void)

Gets the current tilt slew.

• int getBasePanSlew (void)

Gets the base pan slew.

• int getBaseTiltSlew (void)

Gets the base tilt slew.

• int getPanAccel (void)

Gets the current pan acceleration rate.

• int getTiltAccel (void)

Gets the current tilt acceleration rate.

Protected Attributes

• int myPan

adds on extra delim in front to work on H8.

4.37.1 Detailed Description

Driver for the DPPTU.

4.37.2 Member Enumeration Documentation

4.37.2.1 anonymous enum

Enumeration values:

MIN_PAN Maximum pan range of 3090 positions.

 $\mathbf{MAX_TILT}$ Minimum pan range of -3090 positions.

MIN_TILT Maximum tilt range of 600 positions.

MAX_PAN_SLEW Minimum tilt range of -900 positions.

MIN_PAN_SLEW Maximum pan slew of 2902 positions/sec.

MAX_TILT_SLEW Minimum tilt slew of 31 positions/sec.

MIN_TILT_SLEW Maximum tilt slew of 2902 positions/sec.

MAX_PAN_ACCEL Minimum tilt slew of 31 positions/sec.

MIN_PAN_ACCEL Maximum pan acceleration of 2000 positions/sec^2.

MAX_TILT_ACCEL Minimum pan acceleration of 0 positions/ \sec^2 .

MIN_TILT_ACCEL Maximum tilt acceleration of 2000 positions/ \sec^2 2.

4.37.3 Member Function Documentation

4.37.3.1 bool ArDPPTU::blank (void)

Sends a delimiter only.

A blank packet can be sent to exit monitor mode *

- ArDPPTU.h
- ArDPPTU.cpp

4.38 ArDPPTUCommands Class Reference

A class with the commands for the DPPTU.

#include <ArDPPTU.h>

Public Types

enum { DELIM = 0x20, INIT = 0x40, ACCEL = 0x61, BASE = 0x62, CONTROL = 0x63, DISABLE = 0x64, ENABLE = 0x65, FACTORY = 0x66, HALT = 0x68, IMMED = 0x69, LIMIT = 0x6C, MONITOR = 0x6D, OFFSET = 0x6F, PAN = 0x70, RESET = 0x72, SPEED = 0x73, TILT = 0x74, UPPER = 0x75, VELOCITY = 0x76 }

4.38.1 Detailed Description

A class with the commands for the DPPTU.

This class is for controlling the Directed Perceptions Pan-Tilt Unit

Note that there are far too many functions enabled in here, most of which are extraneous. The important ones are defined in the **ArPTZ** (p. 298) class and include the basic pan, tilt commands.

The DPPTU's pan and tilt commands work on a number of units equal to (degrees / 0.514). The panTilt function always rounds the conversion closer to zero, so that a magnitude greater than the allowable range of movement is not sent to the camera.

4.38.2 Member Enumeration Documentation

4.38.2.1 anonymous enum

Enumeration values:

 \mathbf{DELIM} Space - Carriage return delimeter.

INIT Init character.

ACCEL Acceleration, Await position-command completion.

BASE Base speed.

CONTROL Speed control.

DISABLE Disable character, Delta, Default.

ENABLE Enable character, Echoing.

FACTORY Restore factory defaults.

HALT Halt, Hold, High.

IMMED Immediate position-command execution mode, Independent control mode.

LIMIT Position limit character, Low.

MONITOR Monitor, In-motion power mode.

OFFSET Offset position, Off.

PAN Pan.

RESET Reset calibration, Restore stored defaults, Regular.

SPEED Speed, Slave.

TILT Tilt.

 $\mathbf{UPPER}\quad \mathrm{Upper\ speed\ limit}.$

VELOCITY Velocity control mode.

The documentation for this class was generated from the following file:

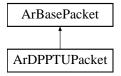
• ArDPPTU.h

4.39 ArDPPTUPacket Class Reference

A class for for making commands to send to the DPPTU.

#include <ArDPPTU.h>

Inheritance diagram for ArDPPTUPacket::



Public Methods

 $\bullet \ \mathbf{ArDPPTUPacket} \ (\mathbf{ArTypes::} \mathbf{UByte2} \ \mathbf{bufferSize} {=} 30)$

Constructor.

• virtual ~ArDPPTUPacket ()

Destructor.

• virtual void finalizePacket (void)

MakeFinals the packet in preparation for sending, must be done.

4.39.1 Detailed Description

A class for for making commands to send to the DPPTU.

There are only a few functioning ways to put things into this packet, you MUST use these, if you use anything else your commands won't work. You must use byteToBuf and byte2ToBuf.

The documentation for this class was generated from the following files:

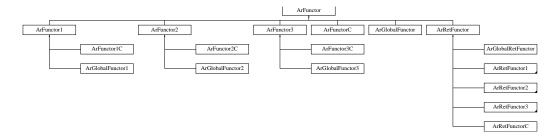
- \bullet ArDPPTU.h
- ArDPPTU.cpp

4.40 ArFunctor Class Reference

Base class for functors.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor::



Public Methods

- virtual ~**ArFunctor** ()

 Destructor.
- virtual void **invoke** (void)=0

 Invokes the functor.

4.40.1 Detailed Description

Base class for functors.

Functors are meant to encapsulate the idea of a pointer to a function which is a member of a class. To use a pointer to a member function, you must have a C style function pointer, 'void(Class::*)()', and a pointer to an instance of the class in which the function is a member of. This is because all non-static member functions must have a 'this' pointer. If they dont and if the member function uses any member data or even other member functions it will not work right and most likely crash. This is because the 'this' pointer is not the correct value and is most likely a random uninitialized value. The virtue of static member functions is that they do not require a 'this' pointer to be run. But the compiler will never let you access any member data or functions from within a static member function.

Because of the design of C++ never allowed for encapsulating these two pointers together into one language supported construct, this has to be done by hand. For

conviences sake, there are functors (**ArGlobalFunctor** (p. 168), **ArGlobal-RetFunctor** (p. 182)) which take a pure C style function pointer (a non-member function). This is in case you want to use a functor that refers to a global C style function.

Aria (p. 205) makes use of functors by using them as callback functions. Since Aria (p. 205) is programmed using the object oriented programming paradigm, all the callback functions need to be tied to an object and a particular instance. Thus the need for functors. Most of the use of callbacks simply take an Ar-Functor, which is the base class for all the functors. This class only has the ability to invoke a functor. All the derivitave functors have the ability to invoke the correct function on the correct object.

Because functions have different signatures because they take different types of parameters and have different number of parameters, templates were used to create the functors. These are the base classes for the functors. These classes encapsulate everything except for the class type that the member function is a member of. This allows someone to accept a functor of type **ArFunctor1** (p. 142)<int> which has one parameter of type 'int'. But they never have to know that the function is a member function of class 'SomeUnknownType'. These classes are:

ArFunctor, ArFunctor1 (p. 142), ArFunctor2 (p. 148), ArFunctor3 (p. 155) ArRetFunctor (p. 327), ArRetFunctor1 (p. 328), ArRetFunctor2 (p. 334), ArRetFunctor3 (p. 342)

These 8 functors are the only thing a piece of code that wants a functor will ever need. But these classes are abstract classes and can not be instantiated. On the other side, the piece of code that wants to be called back will need the functor classes that know about the class type. These functors are:

ArFunctorC (p. 165), ArFunctor1C (p. 144), ArFunctor2C (p. 150), ArFunctor3C (p. 157) ArRetFunctorC (p. 352), ArRetFunctor1C (p. 330), ArRetFunctor2C (p. 336), ArRetFunctor3C (p. 345)

These functors are meant to be instantiated and passed of to a piece of code that wants to use them. That piece of code should only know the functor as one of the functor classes without the 'C' in it.

Note that you can create these FunctorC instances with default arguments that are then used when the invoke is called without those arguments... These are quite useful since if you have a class that expects an ArFunctor you can make an **ArFunctor1C** (p. 144) with default arguments and pass it as an ArFunctor... and it will get called with that default argument, this is useful for having multiple functors use the same function with different arguments and results (just takes one functor each). You can see an example of this in the tests/functor-Test.cpp example (in testBase for example).

See the example functor.cpp for a simple example of using functors.

See the test program functortest.cpp for the full use of all the functors. The documentation for this class was generated from the following file:

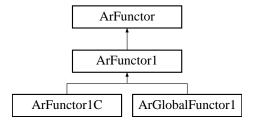
 \bullet ArFunctor.h

4.41 ArFunctor1 Class Template Reference

Base class for functors with 1 parameter.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor1::



Public Methods

• virtual ~ArFunctor1 ()

Destructor.

- virtual void **invoke** (void)=0

 Invokes the functor.
- virtual void **invoke** (P1 p1)=0

 Invokes the functor.

4.41.1 Detailed Description

template<class P1> class ArFunctor1< P1>

Base class for functors with 1 parameter.

This is the base class for functors with 1 parameter. Code that has a reference to a functor that takes 1 parameter should use this class name. This allows the code to know how to invoke the functor without knowing which class the member function is in.

For an overall description of functors, see **ArFunctor** (p. 139).

4.41.2 Member Function Documentation

4.41.2.1 template<class P1> virtual void ArFunctor1< P1 >::invoke (P1 p1) [pure virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented in ArGlobalFunctor1 (p. 171), and ArFunctor1C (p. 146).

The documentation for this class was generated from the following file:

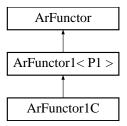
 \bullet ArFunctor.h

4.42 ArFunctor1C Class Template Reference

Functor for a member function with 1 parameter.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor1C::



Public Methods

• ArFunctor1C ()

Constructor.

- ArFunctor1C (T &obj, void(T::*func)(P1))

 Constructor supply function pointer.
- ArFunctor1C (T &obj, void(T::*func)(P1), P1 p1)

 Constructor supply function pointer, default parameters.
- ArFunctor1C (T *obj, void(T::*func)(P1))

 Constructor supply function pointer.
- ArFunctor1C (T *obj, void(T::*func)(P1), P1 p1)

 Constructor supply function pointer, default parameters.
- virtual ~**ArFunctor1C** ()

 Destructor.
- virtual void **invoke** (void)

 Invokes the functor.
- virtual void **invoke** (P1 p1)

 Invokes the functor.
- virtual void **setThis** (T *obj)

Set the 'this' pointer.

- virtual void **setThis** (T &obj)

 Set the 'this' pointer.
- virtual void **setP1** (P1 p1)

 Set the default parameter.

4.42.1 Detailed Description

 $template < class \ T, \ class \ P1 > \ class \ ArFunctor1C < \ T, \ P1 >$

Functor for a member function with 1 parameter.

This is a class for member functions which take 1 parameter. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.42.2 Constructor & Destructor Documentation

4.42.2.1 template<class T, class P1> ArFunctor1C< T, P1 >::ArFunctor1C (T & obj, void(T::* func)(P1)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.42.2.2 template < class T, class P1> ArFunctor1C < T, P1 >::ArFunctor1C (T & obj, void(T::* func)(P1), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.42.2.3 template<class T, class P1> ArFunctor1C< T, P1 >::ArFunctor1C (T * obj, void(T::* func)(P1)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.42.2.4 template < class T, class P1> ArFunctor1C< T, P1 >::ArFunctor1C (T * obj, void(T::* func)(P1), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer p1 default first parameter

4.42.3 Member Function Documentation

4.42.3.1 template<class T, class P1> virtual void ArFunctor1C< T, P1>::invoke (P1 p1) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented from **ArFunctor1** (p. 143).

4.42.3.2 template<class T, class P1> virtual void ArFunctor1C< T, P1>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.42.3.3 template<class T, class P1> virtual void ArFunctor1C< T, P1>::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.42.3.4 template < class T, class P1> virtual void ArFunctor1C< T, P1>::setThis (T * obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

 ${\it obj}$ the 'this' pointer

The documentation for this class was generated from the following file:

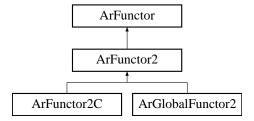
• ArFunctor.h

4.43 ArFunctor2 Class Template Reference

Base class for functors with 2 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor2::



Public Methods

- virtual ~**ArFunctor2** ()

 Destructor.
- virtual void **invoke** (void)=0

 Invokes the functor.
- virtual void **invoke** (P1 p1)=0

 Invokes the functor.
- virtual void **invoke** (P1 p1, P2 p2)=0 *Invokes the functor*.

4.43.1 Detailed Description

template<class P1, class P2> class ArFunctor2< P1, P2>

Base class for functors with 2 parameters.

This is the base class for functors with 2 parameters. Code that has a reference to a functor that takes 2 parameters should use this class name. This allows the code to know how to invoke the functor without knowing which class the member function is in.

For an overall description of functors, see **ArFunctor** (p. 139).

4.43.2 Member Function Documentation

4.43.2.1 template<class P1, class P2> virtual void ArFunctor2< P1, P2>::invoke (P1 p1, P2 p2) [pure virtual]

Invokes the functor.

Parameters:

 $\boldsymbol{p1}$ first parameter

p2 second parameter

Reimplemented in ArGlobalFunctor2 (p. 175), and ArFunctor2C (p. 153).

4.43.2.2 template<class P1, class P2> virtual void ArFunctor2< P1, P2>::invoke (P1 p1) [pure virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented in ArGlobalFunctor2 (p. 175), and ArFunctor2C (p. 153).

The documentation for this class was generated from the following file:

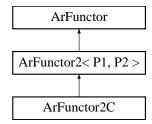
• ArFunctor.h

4.44 ArFunctor2C Class Template Reference

Functor for a member function with 2 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor2C::



Public Methods

 \bullet ArFunctor2C ()

Constructor.

- ArFunctor2C (T &obj, void(T::*func)(P1, P2))

 Constructor supply function pointer.
- ArFunctor2C (T &obj, void(T::*func)(P1, P2), P1 p1)

 Constructor supply function pointer, default parameters.
- ArFunctor2C (T &obj, void(T::*func)(P1, P2), P1 p1, P2 p2)

 Constructor supply function pointer, default parameters.
- **ArFunctor2C** (T *obj, void(T::*func)(P1, P2))

 Constructor supply function pointer.
- ArFunctor2C (T *obj, void(T::*func)(P1, P2), P1 p1)

 Constructor supply function pointer, default parameters.
- ArFunctor2C (T *obj, void(T::*func)(P1, P2), P1 p1, P2 p2)

 Constructor supply function pointer, default parameters.
- virtual ~**ArFunctor2C** ()

 Destructor.
- virtual void **invoke** (void)

Invokes the functor.

- virtual void **invoke** (P1 p1)

 Invokes the functor.
- virtual void **invoke** (P1 p1, P2 p2)

 Invokes the functor.
- virtual void **setThis** (T *obj)

 Set the 'this' pointer.
- virtual void **setThis** (T &obj)

 Set the 'this' pointer.
- virtual void **setP1** (P1 p1)

 Set the default parameter.
- virtual void **setP2** (P2 p2)

 Set the default 2nd parameter.

4.44.1 Detailed Description

template<class T, class P1, class P2> class ArFunctor2C< T, P1, P2 >

Functor for a member function with 2 parameters.

This is a class for member functions which take 2 parameters. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.44.2 Constructor & Destructor Documentation

4.44.2.1 template<class T, class P1, class P2> ArFunctor2C< T, P1, P2>::ArFunctor2C (T & obj, void(T::* func)(P1, P2)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.44.2.2 template<class T, class P1, class P2> ArFunctor2C< T, P1, P2>::ArFunctor2C (T & obj, void(T::* func)(P1, P2), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.44.2.3 template<class T, class P1, class P2> ArFunctor2C< T, P1, P2>::ArFunctor2C (T & obj, void(T::* func)(P1, P2), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer p1 default first parameter p2 default second parameter

4.44.2.4 template<class T, class P1, class P2> ArFunctor2C< T, P1, P2>::ArFunctor2C (T * obj, void(T::* func)(P1, P2))
[inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.44.2.5 template<class T, class P1, class P2> ArFunctor2C< T, P1, P2>::ArFunctor2C (T * obj, void(T::* func)(P1, P2), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.44.2.6 template<class T, class P1, class P2> ArFunctor2C< T, P1, P2>::ArFunctor2C (T * obj, void(T::* func)(P1, P2), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointerp1 default first parameterp2 default second parameter

4.44.3 Member Function Documentation

4.44.3.1 template<class T, class P1, class P2> virtual void ArFunctor2C< T, P1, P2>::invoke (P1 p1, P2 p2) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameterp2 second parameter

Reimplemented from **ArFunctor2** (p. 149).

4.44.3.2 template<class T, class P1, class P2> virtual void ArFunctor2C< T, P1, P2>::invoke (P1 p1) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented from ArFunctor2 (p. 149).

4.44.3.3 template<class T, class P1, class P2> virtual void ArFunctor2C< T, P1, P2>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.44.3.4 template<class T, class P1, class P2> virtual void ArFunctor2C< T, P1, P2>::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

4.44.3.5 template<class T, class P1, class P2> virtual void ArFunctor2C< T, P1, P2>::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.44.3.6 template<class T, class P1, class P2> virtual void ArFunctor2C< T, P1, P2>::setThis (T*obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

The documentation for this class was generated from the following file:

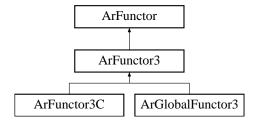
• ArFunctor.h

4.45 ArFunctor3 Class Template Reference

Base class for functors with 3 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor3::



Public Methods

• virtual ~**ArFunctor3** ()

Destructor.

- virtual void **invoke** (void)=0

 Invokes the functor.
- virtual void **invoke** (P1 p1)=0 *Invokes the functor.*
- virtual void **invoke** (P1 p1, P2 p2)=0 *Invokes the functor.*
- virtual void **invoke** (P1 p1, P2 p2, P3 p3)=0 *Invokes the functor.*

4.45.1 Detailed Description

template<class P1, class P2, class P3> class Ar
Functor3< P1, P2, P3 >

Base class for functors with 3 parameters.

This is the base class for functors with 3 parameters. Code that has a reference to a functor that takes 3 parameters should use this class name. This allows

the code to know how to invoke the functor without knowing which class the member function is in.

For an overall description of functors, see **ArFunctor** (p. 139).

4.45.2 Member Function Documentation

4.45.2.1 template<class P1, class P2, class P3> virtual void ArFunctor3< P1, P2, P3>::invoke (P1 p1, P2 p2, P3 p3) [pure virtual]

Invokes the functor.

Parameters:

p1 first parameter

p2 second parameter

p3 third parameter

Reimplemented in ArGlobalFunctor3 (p. 179), and ArFunctor3C (p. 161).

4.45.2.2 template<class P1, class P2, class P3> virtual void ArFunctor3< P1, P2, P3>::invoke (P1 p1, P2 p2) [pure virtual]

Invokes the functor.

Parameters:

p1 first parameter

p2 second parameter

Reimplemented in ArGlobalFunctor3 (p. 180), and ArFunctor3C (p. 161).

4.45.2.3 template<class P1, class P2, class P3> virtual void ArFunctor3< P1, P2, P3>::invoke (P1 p1) [pure virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented in ArGlobalFunctor3 (p. 180), and ArFunctor3C (p. 162).

The documentation for this class was generated from the following file:

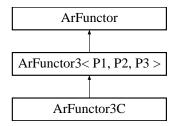
• ArFunctor.h

4.46 ArFunctor3C Class Template Reference

Functor for a member function with 3 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor3C::



Public Methods

• ArFunctor3C ()

Constructor.

• **ArFunctor3C** (T &obj, void(T::*func)(P1, P2, P3))

Constructor - supply function pointer.

 $Constructor\ \hbox{--} \ supply\ function\ pointer,\ default\ parameters.$

• ArFunctor3C (T &obj, void(T::*func)(P1, P2, P3), P1 p1, P2 p2)

Constructor - supply function pointer, default parameters.

• **ArFunctor3C** (T &obj, void(T::*func)(P1, P2, P3), P1 p1, P2 p2, P3 p3)

Constructor - supply function pointer, default parameters.

 $\bullet \ \mathbf{ArFunctor3C} \ (T * obj, \ void(T :: * func)(P1, \ P2, \ P3)) \\$

Constructor - supply function pointer.

• **ArFunctor3C** (T *obj, void(T::*func)(P1, P2, P3), P1 p1)

Constructor - supply function pointer, default parameters.

• ArFunctor3C (T *obj, void(T::*func)(P1, P2, P3), P1 p1, P2 p2)

Constructor - supply function pointer, default parameters.

• **ArFunctor3C** (T *obj, void(T::*func)(P1, P2, P3), P1 p1, P2 p2, P3 p3)

 $Constructor \hbox{ - } supply \hbox{ function pointer, default parameters.}$

• virtual ~ArFunctor3C ()

Destructor.

• virtual void **invoke** (void)

Invokes the functor.

• virtual void **invoke** (P1 p1)

Invokes the functor.

• virtual void invoke (P1 p1, P2 p2)

Invokes the functor.

• virtual void invoke (P1 p1, P2 p2, P3 p3)

Invokes the functor.

• virtual void **setThis** (T *obj)

Set the 'this' pointer.

• virtual void **setThis** (T &obj)

Set the 'this' pointer.

 \bullet virtual void $\mathbf{setP1}$ (P1 p1)

Set the default parameter.

• virtual void setP2 (P2 p2)

Set the default 2nd parameter.

• virtual void setP3 (P3 p3)

Set the default third parameter.

4.46.1 Detailed Description

template<class T, class P1, class P2, class P3> class ArFunctor3C< T, P1, P2, P3 >

Functor for a member function with 3 parameters.

This is a class for member functions which take 3 parameters. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.46.2 Constructor & Destructor Documentation

4.46.2.1 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T & obj, void(T::* func)(P1, P2, P3)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.46.2.2 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T & obj, void(T::* func)(P1, P2, P3), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.46.2.3 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T & obj, void(T::* func)(P1, P2, P3), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

```
func member function pointerp1 default first parameterp2 default second parameter
```

4.46.2.4 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T & obj, void(T::* func)(P1, P2, P3), P1 p1, P2 p2, P3 p3) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter
p2 default second parameter
p3 default third parameter

4.46.2.5 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T * obj, void(T::* func)(P1, P2, P3)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.46.2.6 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T * obj, void(T::* func)(P1, P2, P3), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer p1 default first parameter

4.46.2.7 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T * obj, void(T::* func)(P1, P2, P3), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointerp1 default first parameterp2 default second parameter

4.46.2.8 template<class T, class P1, class P2, class P3> ArFunctor3C< T, P1, P2, P3>::ArFunctor3C (T * obj, void(T::* func)(P1, P2, P3), P1 p1, P2 p2, P3 p3) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

```
func member function pointer
p1 default first parameter
p2 default second parameter
p3 default third parameter
```

4.46.3 Member Function Documentation

4.46.3.1 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::invoke (P1 p1, P2 p2, P3 p3) [inline, virtual]

Invokes the functor.

Parameters:

```
p1 first parameter
```

p2 second parameter

p3 third parameter

Reimplemented from ArFunctor3 (p. 156).

4.46.3.2 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::invoke (P1 p1, P2 p2) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

 $\it p2$ second parameter

Reimplemented from ArFunctor3 (p. 156).

4.46.3.3 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::invoke (P1 p1) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented from ArFunctor3 (p. 156).

4.46.3.4 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.46.3.5 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

4.46.3.6 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::setP3 (P3 p3) [inline, virtual]

Set the default third parameter.

Parameters:

p3 default third parameter

4.46.3.7 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.46.3.8 template<class T, class P1, class P2, class P3> virtual void ArFunctor3C< T, P1, P2, P3>::setThis (T * obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

The documentation for this class was generated from the following file:

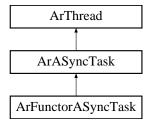
• ArFunctor.h

4.47 ArFunctorASyncTask Class Reference

This is like **ArASyncTask** (p. 110), but instead of runThread it uses a functor to run.

#include <ArFunctorASyncTask.h>

Inheritance diagram for ArFunctorASyncTask::



Public Methods

- ArFunctorASyncTask (ArRetFunctor1< void *, void *> *functor)

 *Constructor.
- virtual ~**ArFunctorASyncTask** ()

 Destructor.
- virtual void * runThread (void *arg)

 Our reimplementation of runThread.

4.47.1 Detailed Description

This is like $\mathbf{ArASyncTask}$ (p. 110), but instead of runThread it uses a functor to run.

The documentation for this class was generated from the following files:

- ArFunctorASyncTask.h
- ArFunctorASyncTask.cpp

4.48 ArFunctorC Class Template Reference

Functor for a member function.

#include <ArFunctor.h>

Inheritance diagram for ArFunctorC::



Public Methods

• ArFunctorC ()

Constructor.

- **ArFunctorC** (T &obj, void(T::*func)(void))

 Constructor supply function pointer.
- **ArFunctorC** (T *obj, void(T::*func)(void))

 Constructor supply function pointer.
- virtual ~ArFunctorC ()

Destructor.

- virtual void **invoke** (void)
 - Invokes the functor.

• virtual void **setThis** (T *obj)

Set the 'this' pointer.

• virtual void **setThis** (T &obj)

Set the 'this' pointer.

4.48.1 Detailed Description

template<class T> class ArFunctorC< T >

Functor for a member function.

This is a class for member functions. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.48.2 Constructor & Destructor Documentation

4.48.2.1 template < class T > ArFunctorC < T >::ArFunctorC (T & obj, void(T::* func)(void)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.48.2.2 template < class T> ArFunctorC< T>:: ArFunctorC (T * obj, void(T::* func)(void)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.48.3 Member Function Documentation

4.48.3.1 template<class T> virtual void ArFunctorC< T>::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.48.3.2 template < class T> virtual void ArFunctorC< T>::setThis (T*obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

The documentation for this class was generated from the following file:

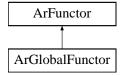
• ArFunctor.h

4.49 ArGlobalFunctor Class Reference

Functor for a global function with no parameters.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalFunctor::



Public Methods

• ArGlobalFunctor ()

Constructor.

• ArGlobalFunctor (void(*func)(void))

 $Constructor \hbox{ - } supply \hbox{ } function \hbox{ } pointer.$

• virtual ~ArGlobalFunctor ()

Destructor.

• virtual void invoke (void)

 $Invokes\ the\ functor.$

4.49.1 Detailed Description

Functor for a global function with no parameters.

This is a class for global functions. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.49.2 Constructor & Destructor Documentation

4.49.2.1 ArGlobalFunctor::ArGlobalFunctor (void(* func)(void)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

The documentation for this class was generated from the following file:

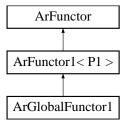
• ArFunctor.h

4.50 ArGlobalFunctor1 Class Template Reference

Functor for a global function with 1 parameter.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalFunctor1::



Public Methods

• ArGlobalFunctor1 ()

Constructor.

• **ArGlobalFunctor1** (void(*func)(P1))

Constructor - supply function pointer.

 $\bullet \ \mathbf{ArGlobalFunctor1} \ (\mathrm{void}(*\mathrm{func})(\mathrm{P1}), \ \mathrm{P1} \ \mathrm{p1}) \\$

Constructor - supply function pointer, default parameters.

• virtual ~ArGlobalFunctor1 ()

Destructor.

• virtual void invoke (void)

Invokes the functor.

• virtual void **invoke** (P1 p1)

Invokes the functor.

• virtual void **setP1** (P1 p1)

Set the default parameter.

4.50.1 Detailed Description

template<class P1> class ArGlobalFunctor1< P1>

Functor for a global function with 1 parameter.

This is a class for global functions which take 1 parameter. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.50.2 Constructor & Destructor Documentation

4.50.2.1 template < class P1> ArGlobalFunctor1 < P1 >::ArGlobalFunctor1 (void(* func)(P1)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.50.2.2 template<class P1> ArGlobalFunctor1< P1 >::ArGlobalFunctor1 (void(* func)(P1), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

```
func global function pointer
p1 default first parameter
```

4.50.3 Member Function Documentation

4.50.3.1 template < class P1> virtual void ArGlobalFunctor1 < P1 >::invoke (P1 p1) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented from ArFunctor1 (p. 143).

4.50.3.2 template<class P1> virtual void ArGlobalFunctor1< P1 >::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

The documentation for this class was generated from the following file:

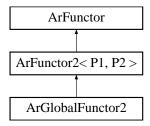
 \bullet Ar
Functor.h

4.51 ArGlobalFunctor2 Class Template Reference

Functor for a global function with 2 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalFunctor2::



Public Methods

• ArGlobalFunctor2 ()

Constructor.

• ArGlobalFunctor2 (void(*func)(P1, P2))

 $Constructor \hbox{ - } supply \hbox{ } function \hbox{ } pointer.$

• ArGlobalFunctor2 (void(*func)(P1, P2), P1 p1)

Constructor - supply function pointer, default parameters.

• ArGlobalFunctor2 (void(*func)(P1, P2), P1 p1, P2 p2)

Constructor - supply function pointer, default parameters.

• virtual ~ArGlobalFunctor2 ()

Destructor.

• virtual void **invoke** (void)

Invokes the functor.

• virtual void **invoke** (P1 p1)

Invokes the functor.

• virtual void invoke (P1 p1, P2 p2)

Invokes the functor.

- virtual void **setP1** (P1 p1)

 Set the default parameter.
- virtual void **setP2** (P2 p2)

 Set the default 2nd parameter.

4.51.1 Detailed Description

template<class P1, class P2> class ArGlobalFunctor2< P1, P2>

Functor for a global function with 2 parameters.

This is a class for global functions which take 2 parameters. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.51.2 Constructor & Destructor Documentation

4.51.2.1 template<class P1, class P2> ArGlobalFunctor2< P1, P2 >::ArGlobalFunctor2 (void(* func)(P1, P2)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.51.2.2 template < class P1, class P2> ArGlobalFunctor2< P1, P2 >::ArGlobalFunctor2 (void(* func)(P1, P2), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointer
p1 default first parameter

4.51.2.3 template < class P1, class P2> ArGlobalFunctor2< P1, P2 >::ArGlobalFunctor2 (void(* func)(P1, P2), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

```
func global function pointerp1 default first parameterp2 default second parameter
```

4.51.3 Member Function Documentation

4.51.3.1 template<class P1, class P2> virtual void ArGlobalFunctor2< P1, P2>::invoke (P1 p1, P2 p2) [inline, virtual]

Invokes the functor.

Parameters:

```
p1 first parameterp2 second parameter
```

Reimplemented from **ArFunctor2** (p. 149).

4.51.3.2 template<class P1, class P2> virtual void ArGlobalFunctor2< P1, P2>::invoke (P1 p1) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented from ArFunctor2 (p. 149).

Set the default parameter.

Parameters:

p1 default first parameter

4.51.3.4 template<class P1, class P2> virtual void ArGlobalFunctor2< P1, P2>::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

The documentation for this class was generated from the following file:

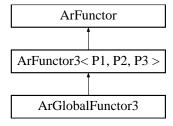
 \bullet Ar
Functor.h

4.52 ArGlobalFunctor3 Class Template Reference

Functor for a global function with 3 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalFunctor3::



Public Methods

• ArGlobalFunctor3 ()

Constructor.

• ArGlobalFunctor3 (void(*func)(P1, P2, P3))

 $Constructor \hbox{ - } supply \hbox{ function pointer}.$

• ArGlobalFunctor3 (void(*func)(P1, P2, P3), P1 p1)

Constructor - supply function pointer, default parameters.

• ArGlobalFunctor3 (void(*func)(P1, P2, P3), P1 p1, P2 p2)

Constructor - supply function pointer, default parameters.

• ArGlobalFunctor3 (void(*func)(P1, P2, P3), P1 p1, P2 p2, P3 p3)

Constructor - supply function pointer, default parameters.

• virtual ~ArGlobalFunctor3 ()

Destructor.

• virtual void **invoke** (void)

Invokes the functor.

• virtual void **invoke** (P1 p1)

Invokes the functor.

- virtual void **invoke** (P1 p1, P2 p2)

 Invokes the functor.
- virtual void **invoke** (P1 p1, P2 p2, P3 p3)

 Invokes the functor.
- virtual void **setP1** (P1 p1)

 Set the default parameter.
- virtual void **setP2** (P2 p2)

 Set the default 2nd parameter.
- virtual void **setP3** (P3 p3)

 Set the default third parameter.

4.52.1 Detailed Description

template<class P1, class P2, class P3> class ArGlobal Functor3< P1, P2, P3 >

Functor for a global function with 3 parameters.

This is a class for global functions which take 3 parameters. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.52.2 Constructor & Destructor Documentation

4.52.2.1 template<class P1, class P2, class P3> ArGlobalFunctor3

P1, P2, P3>::ArGlobalFunctor3 (void(* func)(P1, P2, P3)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.52.2.2 template < class P1, class P2, class P3> ArGlobalFunctor3 < P1, P2, P3 >::ArGlobalFunctor3 (void(* func)(P1, P2, P3), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointer
p1 default first parameter

4.52.2.3 template<class P1, class P2, class P3> ArGlobalFunctor3

P1, P2, P3>::ArGlobalFunctor3 (void(* func)(P1, P2, P3), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointerp1 default first parameterp2 default second parameter

4.52.2.4 template<class P1, class P2, class P3> ArGlobalFunctor3< P1, P2, P3>::ArGlobalFunctor3 (void(* func)(P1, P2, P3), P1 p1, P2 p2, P3 p3) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointer
p1 default first parameter
p2 default second parameter
p3 default third parameter

4.52.3 Member Function Documentation

4.52.3.1 template<class P1, class P2, class P3> virtual void ArGlobalFunctor3< P1, P2, P3>::invoke (P1 p1, P2 p2, P3 p3) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

p2 second parameter

p3 third parameter

Reimplemented from **ArFunctor3** (p. 156).

4.52.3.2 template<class P1, class P2, class P3> virtual void ArGlobalFunctor3< P1, P2, P3>::invoke (P1 p1, P2 p2) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

p2 second parameter

Reimplemented from ArFunctor3 (p. 156).

4.52.3.3 template<class P1, class P2, class P3> virtual void ArGlobalFunctor3< P1, P2, P3>::invoke (P1 p1) [inline, virtual]

Invokes the functor.

Parameters:

p1 first parameter

Reimplemented from **ArFunctor3** (p. 156).

4.52.3.4 template<class P1, class P2, class P3> virtual void ArGlobalFunctor3< P1, P2, P3>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.52.3.5 template<class P1, class P2, class P3> virtual void ArGlobalFunctor3< P1, P2, P3>::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

4.52.3.6 template<class P1, class P2, class P3> virtual void ArGlobalFunctor3< P1, P2, P3>::setP3 (P3 p3) [inline, virtual]

Set the default third parameter.

Parameters:

p3 default third parameter

The documentation for this class was generated from the following file:

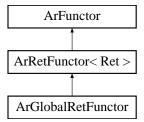
• ArFunctor.h

4.53 ArGlobalRetFunctor Class Template Reference

Functor for a global function with return value.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalRetFunctor::



Public Methods

• ArGlobalRetFunctor ()

Constructor.

 $\bullet \ \mathbf{ArGlobalRetFunctor} \ (\mathrm{Ret}(*\mathrm{func})(\mathrm{void})) \\$

Constructor - supply function pointer.

• virtual ~ArGlobalRetFunctor ()

Destructor.

• virtual Ret invokeR (void)

Invokes the functor with return value.

4.53.1 Detailed Description

template < class Ret > class ArGlobalRetFunctor < Ret >

Functor for a global function with return value.

This is a class for global functions which return a value. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.53.2 Constructor & Destructor Documentation

4.53.2.1 template<class Ret> ArGlobalRetFunctor< Ret >::ArGlobalRetFunctor (Ret(* func)(void)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

The documentation for this class was generated from the following file:

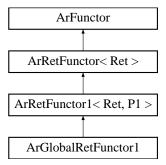
• ArFunctor.h

4.54 ArGlobalRetFunctor1 Class Template Reference

Functor for a global function with 1 parameter and return value.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalRetFunctor1::



Public Methods

• ArGlobalRetFunctor1 ()

Constructor.

• ArGlobalRetFunctor1 (Ret(*func)(P1))

 $Constructor \hbox{ - } supply \hbox{ function pointer}.$

• ArGlobalRetFunctor1 (Ret(*func)(P1), P1 p1)

Constructor - supply function pointer, default parameters.

• virtual ~ArGlobalRetFunctor1 ()

Destructor.

• virtual Ret invokeR (void)

Invokes the functor with return value.

• virtual Ret invokeR (P1 p1)

Invokes the functor with return value.

• virtual void **setP1** (P1 p1)

Set the default parameter.

4.54.1 Detailed Description

template<class Ret, class P1> class ArGlobalRetFunctor1< Ret, P1 >

Functor for a global function with 1 parameter and return value.

This is a class for global functions which take 1 parameter and return a value. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.54.2 Constructor & Destructor Documentation

4.54.2.1 template < class Ret, class P1> ArGlobalRetFunctor1 < Ret, P1>::ArGlobalRetFunctor1 (Ret(* func)(P1)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.54.2.2 template<class Ret, class P1> ArGlobalRetFunctor1< Ret, P1>::ArGlobalRetFunctor1 (Ret(* func)(P1), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointer p1 default first parameter

4.54.3 Member Function Documentation

4.54.3.1 template < class Ret, class P1> virtual Ret ArGlobalRetFunctor1 < Ret, P1>::invokeR (P1 p1) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented from ArRetFunctor1 (p. 329).

4.54.3.2 template<class Ret, class P1> virtual void ArGlobalRetFunctor1< Ret, P1>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

The documentation for this class was generated from the following file:

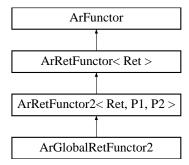
• ArFunctor.h

4.55 ArGlobalRetFunctor2 Class Template Reference

Functor for a global function with 2 parameters and return value.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalRetFunctor2::



Public Methods

• ArGlobalRetFunctor2 ()

Constructor.

• ArGlobalRetFunctor2 (Ret(*func)(P1, P2))

Constructor - supply function pointer.

• ArGlobalRetFunctor2 (Ret(*func)(P1, P2), P1 p1)

Constructor - supply function pointer, default parameters.

• ArGlobalRetFunctor2 (Ret(*func)(P1, P2), P1 p1, P2 p2)

Constructor - supply function pointer, default parameters.

• virtual ~ArGlobalRetFunctor2 ()

Destructor.

• virtual Ret invokeR (void)

Invokes the functor with return value.

• virtual Ret invokeR (P1 p1)

Invokes the functor with return value.

- virtual Ret **invokeR** (P1 p1, P2 p2)

 Invokes the functor with return value.
- virtual void **setP1** (P1 p1) Set the default parameter.
- virtual void **setP2** (P2 p2)

 Set the default 2nd parameter.

4.55.1 Detailed Description

template<class Ret, class P1, class P2> class ArGlobalRetFunctor2< Ret, P1, P2 >

Functor for a global function with 2 parameters and return value.

This is a class for global functions which take 2 parameters and return a value. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.55.2 Constructor & Destructor Documentation

4.55.2.1 template<class Ret, class P1, class P2> ArGlobalRet-Functor2< Ret, P1, P2>::ArGlobalRetFunctor2 (Ret(* func)(P1, P2)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.55.2.2 template<class Ret, class P1, class P2> ArGlobalRet-Functor2< Ret, P1, P2>::ArGlobalRetFunctor2 (Ret(* func)(P1, P2), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointer
p1 default first parameter

4.55.2.3 template<class Ret, class P1, class P2> ArGlobalRet-Functor2< Ret, P1, P2>::ArGlobalRetFunctor2 (Ret(* func)(P1, P2), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointerp2 default second parameter

4.55.3 Member Function Documentation

4.55.3.1 template<class Ret, class P1, class P2> virtual Ret ArGlobalRetFunctor2< Ret, P1, P2>::invokeR (P1 p1, P2 p2) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameterp2 second parameter

Reimplemented from **ArRetFunctor2** (p. 335).

4.55.3.2 template < class Ret, class P1, class P2> virtual Ret ArGlobalRetFunctor2 < Ret, P1, P2 >::invokeR (P1 p1) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented from **ArRetFunctor2** (p. 335).

4.55.3.3 template<class Ret, class P1, class P2> virtual void ArGlobalRetFunctor2< Ret, P1, P2>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.55.3.4 template<class Ret, class P1, class P2> virtual void ArGlobalRetFunctor2< Ret, P1, P2>::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

The documentation for this class was generated from the following file:

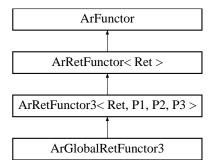
 \bullet Ar
Functor.h

4.56 ArGlobalRetFunctor3 Class Template Reference

Functor for a global function with 2 parameters and return value.

#include <ArFunctor.h>

Inheritance diagram for ArGlobalRetFunctor3::



Public Methods

• ArGlobalRetFunctor3 ()

Constructor.

 $\bullet \ \mathbf{ArGlobalRetFunctor3} \ (\mathrm{Ret}(*\mathrm{func})(\mathrm{P1},\,\mathrm{P2},\,\mathrm{P3})) \\$

Constructor - supply function pointer.

• ArGlobalRetFunctor3 (Ret(*func)(P1, P2, P3), P1 p1)

Constructor - supply function pointer, default parameters.

• ArGlobalRetFunctor3 (Ret(*func)(P1, P2, P3), P1 p1, P2 p2)

Constructor - supply function pointer, default parameters.

• **ArGlobalRetFunctor3** (Ret(*func)(P1, P2, P3), P1 p1, P2 p2, P3 p3)

 $Constructor \hbox{ - } supply \hbox{ function pointer, default parameters.}$

• virtual ~ArGlobalRetFunctor3 ()

Destructor.

• virtual Ret invokeR (void)

Invokes the functor with return value.

- virtual Ret invokeR (P1 p1)
 Invokes the functor with return value.
- virtual Ret **invokeR** (P1 p1, P2 p2)

 Invokes the functor with return value.
- virtual Ret **invokeR** (P1 p1, P2 p2, P3 p3)

 Invokes the functor with return value.
- virtual void **setP1** (P1 p1)

 Set the default parameter.
- virtual void **setP2** (P2 p2)

 Set the default 2nd parameter.
- virtual void **setP3** (P2 p3)

 Set the default third parameter.

4.56.1 Detailed Description

template<class Ret, class P1, class P2, class P3> class ArGlobal Ret-Functor3< Ret, P1, P2, P3 >

Functor for a global function with 2 parameters and return value.

This is a class for global functions which take 2 parameters and return a value. This ties a C style function pointer into the functor class hierarchy as a convience. Code that has a reference to this class and treat it as an **ArFunctor** (p. 139) can use it like any other functor.

For an overall description of functors, see **ArFunctor** (p. 139).

4.56.2 Constructor & Destructor Documentation

4.56.2.1 template < class Ret, class P1, class P2, class P3> ArGlobalRetFunctor3 < Ret, P1, P2, P3 >::ArGlobalRetFunctor3 (Ret(* func)(P1, P2, P3)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.56.2.2 template<class Ret, class P1, class P2, class P3> ArGlobalRetFunctor3< Ret, P1, P2, P3 >::ArGlobalRetFunctor3 (Ret(* func)(P1, P2, P3), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointer p1 default first parameter

4.56.2.3 template < class Ret, class P1, class P2, class P3> ArGlobalRetFunctor3 < Ret, P1, P2, P3 >::ArGlobalRetFunctor3 (Ret(* func)(P1, P2, P3), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointerp1 default first parameterp2 default second parameter

4.56.2.4 template < class Ret, class P1, class P2, class P3> ArGlobalRetFunctor3 < Ret, P1, P2, P3 >::ArGlobalRetFunctor3 (Ret(* func)(P1, P2, P3), P1 p1, P2 p2, P3 p3) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func global function pointerp1 default first parameterp2 default second parameter

4.56.3 Member Function Documentation

4.56.3.1 template<class Ret, class P1, class P2, class P3> virtual Ret ArGlobalRetFunctor3< Ret, P1, P2, P3>::invokeR (P1 p1, P2 p2, P3 p3) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

p2 second parameter

p3 third parameter

Reimplemented from **ArRetFunctor3** (p. 343).

4.56.3.2 template<class Ret, class P1, class P2, class P3> virtual Ret ArGlobalRetFunctor3< Ret, P1, P2, P3>::invokeR (P1 p1, P2 p2) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

p2 second parameter

Reimplemented from **ArRetFunctor3** (p. 343).

4.56.3.3 template<class Ret, class P1, class P2, class P3> virtual Ret ArGlobalRetFunctor3< Ret, P1, P2, P3>::invokeR (P1 p1) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented from **ArRetFunctor3** (p. 343).

4.56.3.4 template<class Ret, class P1, class P2, class P3> virtual void ArGlobalRetFunctor3< Ret, P1, P2, P3>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.56.3.5 template<class Ret, class P1, class P2, class P3> virtual void ArGlobalRetFunctor3< Ret, P1, P2, P3>::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

4.56.3.6 template<class Ret, class P1, class P2, class P3> virtual void ArGlobalRetFunctor3< Ret, P1, P2, P3>::setP3 (P2 p3) [inline, virtual]

Set the default third parameter.

Parameters:

p3 default third parameter

The documentation for this class was generated from the following file:

• ArFunctor.h

4.57 ArGripper Class Reference

A class of convenience functions for using the gripper.

#include <ArGripper.h>

Public Types

• enum Type { QUERYTYPE, GENIO, USERIO, GRIPPAC, NO-GRIPPER }

These are the types for the gripper.

Public Methods

- **ArGripper** (**ArRobot** *robot, int gripperType=QUERYTYPE)

 Constructor.
- virtual ~**ArGripper** ()

 Destructor.
- bool **gripOpen** (void)

 Opens the gripper paddles.
- bool **gripClose** (void)

 Closes the gripper paddles.
- bool **gripStop** (void)

 Stops the gripper paddles.
- bool **liftUp** (void)

 Raises the lift to the top.
- bool **liftDown** (void)

 Lowers the lift to the bottom.
- bool **liftStop** (void)

 Stops the lift.
- ullet bool **gripperStore** (void)

Puts the gripper in a storage position.

• bool **gripperDeploy** (void)

Puts the gripper in a deployed position, ready for use.

• bool **gripperHalt** (void)

Halts the lift and the gripper paddles.

• bool **gripPressure** (int mSecIntervals)

Sets the amount of pressure the gripper applies.

• bool **liftCarry** (int mSecIntervals)

Raises the lift by a given amount of time.

• bool **isGripMoving** (void) const

Returns true if the gripper paddles are moving.

• bool isLiftMoving (void) const

Returns true if the lift is moving.

• int getGripState (void) const

Returns the state of the gripper paddles.

• int **getPaddleState** (void) const

Returns the state of each gripper paddle.

• int getBreakBeamState (void) const

Returns the state of the gripper's breakbeams.

• bool isLiftMaxed (void) const

Returns the state of the lift.

• int **getType** (void) const

Gets the type of the gripper.

• void **setType** (int type)

Sets the type of the gripper.

• long getMSecSinceLastPacket (void) const

Gets the number of mSec since the last gripper packet.

• int **getGraspTime** (void) const

Gets the grasp time.

 $\bullet \ \mathrm{void} \ \mathbf{logState} \ (\mathrm{void}) \ \mathrm{const}$

logs the gripper state.

• bool packetHandler (ArRobotPacket *packet)

Parses the gripper packet.

• void connectHandler (void)

The handler for when the robot connects.

4.57.1 Detailed Description

A class of convenience functions for using the gripper.

The commands which start with grip are for the gripper paddles, the ones which start with lift are the for the lift, and the ones which start with gripper are for the entire unit.

4.57.2 Member Enumeration Documentation

4.57.2.1 enum ArGripper::Type

These are the types for the gripper.

Enumeration values:

QUERYTYPE Finds out what type from the robot, default.

GENIO Uses general IO.

USERIO Uses the user IO.

GRIPPAC Uses a packet requested from the robot.

NOGRIPPER There isn't a gripper.

4.57.3 Constructor & Destructor Documentation

4.57.3.1 ArGripper::ArGripper (ArRobot * robot, int gripperType = QUERYTYPE)

Constructor.

Parameters:

robot The robot this gripper is attached to

useGenIO Whether the gripper on this robot is using GenIO or not

4.57.4 Member Function Documentation

4.57.4.1 int ArGripper::getBreakBeamState (void) const

Returns the state of the gripper's breakbeams.

Returns:

0 if no breakbeams broken, 1 if inner breakbeam broken, 2 if outter breakbeam broken, 3 if both breakbeams broken

4.57.4.2 int ArGripper::getGraspTime (void) const

Gets the grasp time.

If you are using this as anything other than GRIPPAC and you want to find out the grasp time again, just do a setType with QUERYTYPE and it will query the robot again and get the grasp time from the robot.

Returns:

the number of 20 MSec intervals the gripper will continue grasping for after both paddles are triggered

4.57.4.3 int ArGripper::getGripState (void) const

Returns the state of the gripper paddles.

Returns:

0 if gripper paddles between open and closed, 1 if gripper paddles are open, 2 if gripper paddles are closed

4.57.4.4 long ArGripper::getMSecSinceLastPacket (void) const

Gets the number of mSec since the last gripper packet.

Returns:

the number of milliseconds since the last packet

4.57.4.5 int ArGripper::getPaddleState (void) const

Returns the state of each gripper paddle.

Returns:

0 if no gripper paddles are triggered, 1 if the left paddle is triggered, 2 if the right paddle is triggered, 3 if both are triggered

4.57.4.6 int ArGripper::getType (void) const

Gets the type of the gripper.

Returns:

the gripper type

See also:

Type (p. 198)

4.57.4.7 bool ArGripper::gripClose (void)

Closes the gripper paddles.

Returns:

whether the command was sent to the robot or not

4.57.4.8 bool ArGripper::gripOpen (void)

Opens the gripper paddles.

Returns:

whether the command was sent to the robot or not

4.57.4.9 bool ArGripper::gripPressure (int mSecIntervals)

Sets the amount of pressure the gripper applies.

Returns:

whether the command was sent to the robot or not

4.57.4.10 bool ArGripper::gripStop (void)

Stops the gripper paddles.

Returns:

whether the command was sent to the robot or not

4.57.4.11 bool ArGripper::gripperDeploy (void)

Puts the gripper in a deployed position, ready for use.

Returns:

whether the command was sent to the robot or not

4.57.4.12 bool ArGripper::gripperHalt (void)

Halts the lift and the gripper paddles.

Returns:

whether the command was sent to the robot or not

4.57.4.13 bool ArGripper::gripperStore (void)

Puts the gripper in a storage position.

Returns:

whether the command was sent to the robot or not

4.57.4.14 bool ArGripper::isGripMoving (void) const

Returns true if the gripper paddles are moving.

Returns:

true if the gripper paddles are moving

4.57.4.15 bool ArGripper::isLiftMaxed (void) const

Returns the state of the lift.

Returns:

false if lift is between up and down, true is either all the way up or down

4.57.4.16 bool ArGripper::isLiftMoving (void) const

Returns true if the lift is moving.

Returns:

true if the lift is moving

4.57.4.17 bool ArGripper::liftCarry (int mSecIntervals)

Raises the lift by a given amount of time.

Returns:

whether the command was sent to the robot or not

4.57.4.18 bool ArGripper::liftDown (void)

Lowers the lift to the bottom.

Returns:

whether the command was sent to the robot or not

4.57.4.19 bool ArGripper::liftStop (void)

Stops the lift.

Returns:

whether the command was sent to the robot or not

4.57.4.20 bool ArGripper::liftUp (void)

Raises the lift to the top.

Returns:

whether the command was sent to the robot or not

4.57.4.21 void ArGripper::setType (int type)

Sets the type of the gripper.

Parameters:

type the type of gripper to set it to

The documentation for this class was generated from the following files:

- ArGripper.h
- ArGripper.cpp

4.58 ArGripperCommands Class Reference

A class with an enum of the commands for the gripper.

#include <ArGripper.h>

Public Types

enum Commands { GRIP_OPEN = 1, GRIP_CLOSE = 2, GRIP_STOP = 3, LIFT_UP = 4, LIFT_DOWN = 5, LIFT_STOP = 6, GRIPPER_STORE = 7, GRIPPER_DEPLOY = 8, GRIPPER_HALT = 15, GRIP_PRESSURE = 16, LIFT_CARRY }

4.58.1 Detailed Description

A class with an enum of the commands for the gripper.

A class with an enum of the commands for the gripper, see the p2 operations manual and the gripper guide for more detailed descriptions. The enum values which start with GRIP are for the gripper paddles, the ones which start with LIFT are the for the lift, and the ones which start with GRIPPER are for the entire unit.

4.58.2 Member Enumeration Documentation

4.58.2.1 enum ArGripperCommands::Commands

Enumeration values:

GRIP_OPEN open the gripper paddles fully.

GRIP_CLOSE close the gripper paddles all the way.

GRIP_STOP stop the gripper paddles where they are.

LIFT_UP raises the lift to the top of its range.

LIFT_DOWN lowers the lift to the bottom of its range.

LIFT_STOP stops the lift where it is.

GRIPPER_STORE closes the paddles and raises the lift simultaneously, this is for storage not for grasping/carrying an object

GRIPPER_DEPLOY opens the paddles and lowers the lieft simultaneously, this is for getting ready to grasp an object, not for object drops

GRIPPER_HALT stops the gripper paddles and lift from moving.

- **GRIP_PRESSURE** sets the time delay in 20 msec increments after the gripper paddles first grasp an object before they stop moving, regulates grasp pressure
- **LIFT_CARRY** raises or lowers the lieft, the argument is the number of 20 msec increments to raise or lower the lift, poseitive arguments for raise, negative for lower

The documentation for this class was generated from the following file:

• ArGripper.h

4.59 Aria Class Reference

This class performs global initialization and deinitialization.

#include <ariaInternal.h>

Public Types

enum SigHandleMethod { SIGHANDLE_SINGLE, SIGHANDLE_THREAD, SIGHANDLE_NONE }

Static Public Methods

• void **init** (**SigHandleMethod** method=SIGHANDLE_SINGLE, bool initSockets=true)

Performs OS-specific initialization.

• void **uninit** ()

Performs OS-specific deinitialization.

• void addInitCallBack (ArFunctor *cb, ArListPos::Pos position)

Adds a callback to call when Aria is inited.

• void addUninitCallBack (ArFunctor *cb, ArListPos::Pos position)

Adds a callback to call when Aria is uninited.

• void **shutdown** ()

Shutdown all Aria processes/threads.

• void exit ()

Force an exit of all Aria processes/threads.

• bool **getRunning** (void)

Sees if Aria is still running (mostly for the thread in main).

• void addRobot (ArRobot *robot)

Add a robot to the global list of robots.

• void delRobot (ArRobot *robot)

Remove a robot from the global list of robots.

- ArRobot * findRobot (char *name)

 Finds a robot in the global list of robots, by name.
- std::list< ArRobot *> * getRobotList ()

 Get a copy of the global robot list.
- void **setDirectory** (const char *directory)

 Sets the directory that ARIA resides in.
- std::string **getDirectory** (void)

 Gets the directory that ARIA resides in.
- void setKeyHandler (ArKeyHandler *keyHandler)
 Sets the key handler, so that other classes can find it.
- ArKeyHandler * getKeyHandler (void)

 Gets the key handler if one has been set.
- void **signalHandlerCB** (int sig)

 Internal, the callback for the signal handling.

4.59.1 Detailed Description

This class performs global initialization and deinitialization.

4.59.2 Member Enumeration Documentation

4.59.2.1 enum Aria::SigHandleMethod

Enumeration values:

SIGHANDLE_SINGLE Setup signal handlers in a global, non-thread way.

SIGHANDLE_THREAD Setup a dedicated signal handling thread. **SIGHANDLE_NONE** Do no signal handling.

4.59.3 Member Function Documentation

4.59.3.1 void Aria::addInitCallBack (ArFunctor * cb, ArListPos::Pos position) [static]

Adds a callback to call when Aria is inited.

This will add a callback to the list of callbacks to call when Aria has been initialized. It can be called before anything else.

4.59.3.2 void Aria::addUninitCallBack (ArFunctor * cb, ArListPos::Pos position) [static]

Adds a callback to call when Aria is uninited.

This will add a callback to the list of callbacks to call right before Aria is uninitialized. It can be called before anything else. This facilitates code that in operating system signal handlers simply calls **Aria::uninit()** (p. 209) and packages that are based on Aria are unitited as well. It simplifies the entire uninit process.

4.59.3.3 void Aria::exit () [static]

Force an exit of all Aria processes/threads.

This calls cancel() on all AtThread's and ArASyncTask (p. 110)'s. It forces each thread to exit and should only be used in the case of a thread hanging or getting stuck in an infinite loop. This works fine in Linux. In Windows it is not recommended at all that this function be called. Windows can not handle cleanly killing off a thread. See the help in the VC++ compiler on the WIN32 function TerminateThread. The biggest problem is that the state of DLL's can be destroyed.

4.59.3.4 ArRobot * Aria::findRobot (char * name) [static]

Finds a robot in the global list of robots, by name.

Parameters:

name the name of the robot you want to find

Returns:

NULL if there is no robot of that name, otherwise the robot with that name

4.59.3.5 std::string Aria::getDirectory (void) [static]

Gets the directory that ARIA resides in.

This gets the directory that ARIA is located in, this is so ARIA can find param files and the like.

Returns:

the directory ARIA is located in

```
See also:
```

setDirectory (p. 208)

4.59.3.6 bool Aria::getRunning (void) [static]

Sees if Aria is still running (mostly for the thread in main).

This returns if the ARIA stuff is running, which is defined as the time between **Aria::init** (p. 208) and any of **Aria::shutdown** (p. 209), **Aria::exit** (p. 207), or the signal handler kicking off.

4.59.3.7 void Aria::init (SigHandleMethod method = SIGHANDLE_SINGLE, bool initSockets = true) [static]

Performs OS-specific initialization.

This must be called first before any other Aria functions. It initializes the thread layer and the signal handling method. For Windows it iniatializes the socket layer as well. This also sets the directory Aria is located in from the ARIA environmental variable, for a description of this see getDirectory and setDirectory.

For Linux the default signal handling method is to cleanly close down the program, cause all the instances of **ArRobot** (p. 355) to stop their run loop and disconnect from their robot. The program will exit on the following signals: SigHUP, SigINT, SigQUIT, and SigTERM.

For Windows, there is no signal handling.

Parameters:

method the method in which to handle signals. Defaulted to SIGHAN-DLE_SINGLE.

initSockets specify whether or not to initialize the socket layer. This is only meaningfull for Windows. Defaulted to true.

See also:

ArSignalHandler (p. 453), ArSocket (p. 461)

4.59.3.8 void Aria::setDirectory (const char * directory) [static]

Sets the directory that ARIA resides in.

This sets the directory that ARIA is located in, so ARIA can find param files and the like. This can also be controlled by the environment variable ARIA, which this is set to (if it exists) when **Aria::init** (p. 208) is done. So for setDirectory to be effective, it must be done after the **Aria::init** (p. 208).

Parameters:

directory the directory Aria is located in

See also:

getDirectory (p. 207)

4.59.3.9 void Aria::shutdown () [static]

Shutdown all Aria processes/threads.

This calls stop() on all **ArThread** (p. 487)'s and **ArASyncTask** (p. 110)'s. It will block until all **ArThread** (p. 487)'s and **ArASyncTask** (p. 110)'s exit. It is expected that all the tasks will obey the **ArThread::myRunning** (p. 489) variable and exit when it is false.

4.59.3.10 void Aria::uninit () [static]

Performs OS-specific deinitialization.

This must be called last, after all other Aria functions. For both Linux and Windows, it closes all the open ArModules. For Windows it deinitializes the socket layer as well.

The documentation for this class was generated from the following files:

- \bullet ariaInternal.h
- Aria.cpp

4.60 ArInterpolation Class Reference

 $\verb"#include" < \verb"ArInterpolation.h">$

Public Methods

• **ArInterpolation** (size_t numberOfReadings=100)

Constructor.

• virtual ~ArInterpolation ()

Destructor.

• bool addReading (ArTime timeOfReading, ArPose position)

Adds a new reading.

• int getPose (ArTime timeStamp, ArPose *position)

Finds a position.

• void **setNumberOfReadings** (size_t numberOfReadings)

Sets the number of readings this instance holds back in time.

• size_t getNumberOfReadings (void) const

Gets the number of readings this instance holds back in time.

• void **reset** (void)

Empties the interpolated positions.

4.60.1 Detailed Description

This class takes care of storing in readings of position vs time, and then interpolating between them to find where the robot was at a particular point in time. It has two lists, one containing the times, and one containing the positions at those same times (per position), they must be walked through jointly to maintain cohesion. The new entries are at the front of the list, while the old ones are at the back. numberOfReadings and the setNumberOfReadings control the number of entries in the list. If a size is set that is smaller than the current size, then the old ones are chopped off.

4.60.2 Member Function Documentation

4.60.2.1 int ArInterpolation::getPose (ArTime timeStamp, ArPose * position)

Finds a position.

Parameters:

timeStamp the time we are interested in position the pose to set to the given position

Returns:

1 its good interpolation, 0 its predicting, -1 its too far to predict, -2 its too old, -3 there's not enough data to predict

The documentation for this class was generated from the following files:

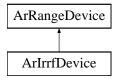
- ArInterpolation.h
- ArInterpolation.cpp

4.61 ArIrrfDevice Class Reference

A class for connecting to a PB-9 and managing the resulting data.

#include <ArIrrfDevice.h>

Inheritance diagram for ArIrrfDevice::



Public Methods

• ArIrrfDevice (size_t currentBufferSize=91, size_t cumulativeBuffer-Size=273, const char *name="irrf")

Constructor.

• ~ArIrrfDevice ()

Destructor.

• bool packetHandler (ArRobotPacket *packet)

The packet handler for use when connecting to an H8 micro-controller.

• void **setCumulativeMaxRange** (double r)

Maximum range for a reading to be added to the cumulative buffer (mm).

• virtual void **setRobot** (**ArRobot** *)

Sets the robot this device is attached to.

4.61.1 Detailed Description

A class for connecting to a PB-9 and managing the resulting data.

This class is for use with a PB9 IR rangefinder. It has the packethandler necessary to process the packets, and will put the data into ArRangeBuffers for use with obstacle avoidance, etc.

The PB9 is still under development, and only works on an H8 controller running AROS.

4.61.2 Member Function Documentation

4.61.2.1 bool ArIrrfDevice::packetHandler (ArRobotPacket * packet)

The packet handler for use when connecting to an H8 micro-controller.

This is the packet handler for the PB9 data, which is sent via the micro controller, to the client. This will read the data from the packets, and then call processReadings to filter add the data to the current and cumulative buffers.

The documentation for this class was generated from the following files:

- ArIrrfDevice.h
- ArIrrfDevice.cpp

4.62 ArJoyHandler Class Reference

Interfaces to a joystick.

#include <ArJoyHandler.h>

Public Methods

- **ArJoyHandler** (bool useOSCal=true, bool useOldJoystick=false) *Constructor.*
- \sim ArJoyHandler ()

Destructor.

• bool init (void)

Intializes the joystick, returns true if successful.

• bool haveJoystick (void)

Returns if the joystick was successfully initialized or not.

- void **getDoubles** (double *x, double *y, double *z=NULL)

 Gets the adjusted reading, as floats, between -1.0 and 1.0.
- bool **getButton** (unsigned int button)

Gets the button.

• bool **haveZAxis** (void)

Returns true if we definitely have a Z axis (we don't know in windows unless it moves).

• void **setSpeeds** (int x, int y, int z=0)

Sets the max that X or Y will return.

- void getAdjusted (int *x, int *y, int *z=NULL)
 Gets the adjusted reading, as integers, based on the setSpeed.
- unsigned int **getNumAxes** (void)

Gets the number of axes the joystick has.

• double **getAxis** (unsigned int axis)

Gets the floating (-1 to 1) location of the given joystick axis.

• unsigned int **getNumButtons** (void)

Gets the number of buttons the joystick has.

• void **setUseOSCal** (bool useOSCal)

Sets whether to just use OS calibration or not.

• bool getUseOSCal (void)

Gets whether to just use OS calibration or not.

• void **startCal** (void)

Starts the calibration process.

• void **endCal** (void)

Ends the calibration process.

• void **getUnfiltered** (int *x, int *y, int *z=NULL)

Gets the unfilitered reading, mostly for internal use, maybe useful for Calibration.

• void **getStats** (int *maxX, int *minX, int *maxY, int *minY, int *cenX, int *cenY)

Gets the stats for the joystick, useful after calibrating to save values.

• void **setStats** (int maxX, int minX, int maxY, int minY, int cenX, int cenY)

Sets the stats for the joystick, useful for restoring calibrated settings.

• void **getSpeeds** (int *x, int *y, int *z)

Gets the speeds that X and Y are set to.

4.62.1 Detailed Description

Interfaces to a joystick.

The joystick handler keeps track of the minimum and maximums for both axes, updating them to constantly be better calibrated. The speeds set influence what is returned by getAdjusted...

The joystick is not opened until init is called. What should basically be done to use this class is to 'init' a joystick, do a 'setSpeed' so you can use 'getAdusted', then at some point do a 'getButton' to see if a button is pressed, and then do a 'getAdjusted' to get the values to act on.

Also note that x is usually rotational velocity (since it right/left), whereas Y is translational (since it is up/down).

You can also use this to do multiple uses with the joystick, for example to have button 1 drive the robot while to have button 2 move the camera, you can get the different values you want (don't want to move the camera as quickly or as far as the robot) by using setSpeed before doing getAdjusted since setSpeed is fast and won't take any time.

4.62.2 Constructor & Destructor Documentation

4.62.2.1 Ar Joy
Handler::Ar Joy Handler (bool useOSCal = true, bool useOld = false)

Constructor.

Parameters:

use OSCal if this is set then the joystick will just rely on the OS to calibrate, otherwise it will keep track of center min and max and use those values for calibration

use Old use the old linux interface to the joystick

4.62.3 Member Function Documentation

4.62.3.1 void ArJoyHandler::endCal (void)

Ends the calibration process.

Ends the calibration, which also sets the center to where the joystick is when the function is called... the center is never reset except in this function, whereas the min and maxes are constantly checked

See also:

startCal (p. 219)

4.62.3.2 void ArJoy Handler::getAdjusted (int * x, int * y, int * z = NULL)

Gets the adjusted reading, as integers, based on the setSpeed.

if useOSCal is true then this returns the readings as calibrated from the OS. If useOSCal is false this finds the percentage of the distance between center and max (or min) then takes this percentage and multiplies it by the speeds given the class, and returns the values computed from this.

Parameters:

 \boldsymbol{x} pointer to an integer in which to store the x value, which is between - x given in set speeds and x given in set speeds

y pointer to an integer in which to store the y value, which is between - y given in set speeds and y given in set speeds

4.62.3.3 double ArJoyHandler::getAxis (unsigned int axis)

Gets the floating (-1 to 1) location of the given joystick axis.

Parameters:

axis axis to get the value of axes are 1 through getNumAxes() (p. 218)

Returns:

true if the button is pressed, false otherwise

4.62.3.4 bool ArJoyHandler::getButton (unsigned int button)

Gets the button.

Parameters:

button button to test for pressed, buttons are 1 through $\mathbf{getNum-Buttons}()$ (p. 218)

Returns:

true if the button is pressed, false otherwise

4.62.3.5 void ArJoyHandler::getDoubles (double * x, double * y, double * z = NULL)

Gets the adjusted reading, as floats, between -1.0 and 1.0.

If useOSCal is true then this returns the readings as calibrated from the OS. If useOSCal is false this finds the percentage of the distance between center and max (or min) then takes this percentage and multiplies it by the speeds given the class, and returns the values computed from this.

Parameters:

- \boldsymbol{x} pointer to a double in which to store the x value, this value is a value between -1.0 and 1.0, for where the stick is on that axis
- y pointer to a double in which to store the y value, this value is a value between -1.0 and 1.0, for where the stick is on that axis

4.62.3.6 unsigned int ArJoyHandler::getNumAxes (void)

Gets the number of axes the joystick has.

Returns:

the number of axes (1 through this number)

4.62.3.7 unsigned int ArJoyHandler::getNumButtons (void)

Gets the number of buttons the joystick has.

Returns:

the number of buttons (1 through this number)

4.62.3.8 void ArJoyHandler::getUnfiltered (int * x, int * y, int * z = NULL)

Gets the unfilitered reading, mostly for internal use, maybe useful for Calibration.

This returns the raw value from the joystick... with X and Y varying between -128 and poseitive 128... this shouldn't be used except in calibration since it'll give very strange readings. For example its not uncommon for a joystick to move 10 to the right but 50 or 100 to the left, so if you aren't adjusting for this you get a robot (or whatever) that goes left really fast, but will hardly go right, hence you should use getAdjusted exclusively except for display in calibration.

Parameters:

- \boldsymbol{x} pointer to an integer in which to store x value
- y pointer to an integer in which to store y value

4.62.3.9 bool ArJoyHandler::getUseOSCal (void)

Gets whether to just use OS calibration or not.

Returns:

if useOSCal is set then the joystick will just rely on the OS to calibrate, otherwise it will keep track of center min and max and use those values for calibration

${\bf 4.62.3.10 \quad void \ Ar Joy Handler::} set Use OSCal \ (bool \ use OSCal)$

Sets whether to just use OS calibration or not.

Parameters:

useOSCal if this is set then the joystick will just rely on the OS to calibrate, otherwise it will keep track of center min and max and use those values for calibration

4.62.3.11 void ArJoyHandler::startCal (void)

Starts the calibration process.

Starts the calibration, which resets all the min and max variables as well as the center variables.

See also:

endCal (p. 216)

The documentation for this class was generated from the following files:

- ArJoyHandler.h
- ArJoyHandler.cpp
- ArJoyHandler_LIN.cpp
- ArJoyHandler_WIN.cpp

4.63 ArKeyHandler Class Reference

This class will read input from the keyboard.

#include <ArKeyHandler.h>

Public Types

• enum KEY { UP = 256, DOWN, LEFT, RIGHT, ESCAPE, SPACE, TAB, ENTER, BACKSPACE, F1, F2, F3, F4 }

These are enums for the non-ascii keys.

Public Methods

• ArKeyHandler (bool blocking=false)

Constructor.

• ~ArKeyHandler ()

Destructor.

• bool addKeyHandler (int keyToHandle, ArFunctor *functor)

This adds a keyhandler, when the keyToHandle is hit, functor will fire.

• bool **remKeyHandler** (int keyToHandler)

This removes a key handler, by key.

• bool remKeyHandler (ArFunctor *functor)

This removes a key handler, by key.

• void **restore** (void)

Sets stdin back to its original settings, if its been restored it won't read anymore.

• void **checkKeys** (void)

intnernal, use addKeyHandler, Checks for keys and handles them.

 \bullet int \mathbf{getKey} (void)

internal, use addKeyHandler instead... Gets a key from the stdin if ones available, -1 if there aren't any available.

4.63.1 Detailed Description

This class will read input from the keyboard.

This class is for handling input from the keyboard, you just addKeyHandler the keys you want to deal with.

You should also register the keyhandler with **Aria::setKeyHandler** (p. 206), and before you create a key handler you should see if one is already there with **Aria::getKeyHandler** (p. 206).

You can attach a key handler to a robot with **ArRobot::attachKeyHandler** (p. 377) which will put a task into the robots list of tasks so that it'll get checked every cycle or you can just call checkKeys yourself (like in its own thread or in the main thread). You should only attach a key handler to one robot, even if you're using multiple robots.

4.63.2 Member Enumeration Documentation

4.63.2.1 enum ArKeyHandler::KEY

These are enums for the non-ascii keys.

```
Enumeration values:
```

```
UP Up arrow (keypad or 4 key dirs).
DOWN Down arrow (keypad or 4 key dirs).
LEFT Left arrow (keypad or 4 key dirs).
RIGHT Right arrow (keypad or 4 key dirs).
ESCAPE Escape key.
SPACE Space key.
TAB Tab key.
ENTER Enter key.
BACKSPACE Backspace key.
F1 F1.
F2 F2.
F3 F3.
F4 F4.
```

4.63.3 Constructor & Destructor Documentation

4.63.3.1 ArKeyHandler::ArKeyHandler (bool blocking = false)

Constructor.

Parameters:

blocking whether or not to block waiting on keys, default is false, ie not to wait... you probably only want to block if you are using check-Keys yourself like after you start a robot run or in its own thread or something along those lines

4.63.4 Member Function Documentation

4.63.4.1 bool ArKeyHandler::addKeyHandler (int keyToHandle, ArFunctor * functor)

This adds a keyhandler, when the keyToHandle is hit, functor will fire.

Parameters:

keyToHandle this is an ascii character, such as 'a' or '1' or '[', or a member of the KEY enum.

functor a functor to call when the given key is pressed

Returns:

true if the addKeyHandler succeeded, which means that the key added was unique and it will be handled... false means that the add failed, because there was already a keyHandler in place for that key

4.63.4.2 bool ArKeyHandler::remKeyHandler (ArFunctor * functor)

This removes a key handler, by key.

Parameters:

keyToHandle the functor to remove

Returns

true if the remKeyHandler succeeded, which means that the key wad found and rmeoved... false means that the remove failed because there was no key for that

4.63.4.3 bool ArKeyHandler::remKeyHandler (int keyToHandle)

This removes a key handler, by key.

Parameters:

keyToHandle this is an ascii character, such as 'a' or '1' or '[', or a member of the KEY enum.

Returns:

true if the remKeyHandler succeeded, which means that the key wad found and rmeoved... false means that the remove failed because there was no key for that

The documentation for this class was generated from the following files:

- \bullet Ar KeyHandler.h
- \bullet ArKeyHandler.cpp

4.64 ArLine Class Reference

This is the class for a line to do some geometric manipulation.

#include <ariaUtil.h>

Public Methods

- **ArLine** (double a, double b, double c)

 Constructor with parameters.
- **ArLine** (double x1, double y1, double x2, double y2)

 Constructor with endpoints.
- virtual ~**ArLine** ()

 Destructor.
- void **newParameters** (double a, double b, double c)

 Sets the line parameters (make it not a segment).
- void **newParametersFromEndpoints** (double x1, double y1, double x2, double y2)

Sets the line parameters from endpoints, but makes it not a segment.

- const double **getA** (void) const Gets the A line parameter.
- const double **getB** (void) const Gets the B line parameter.
- const double **getC** (void) const Gets the C line parameter.
- bool intersects (const ArLine *line, ArPose *pose)

 finds the intersection of this line with another line.
- void makeLinePerp (const ArPose *pose, ArLine *line)

 Makes the given line perpendicular to this one though the given pose.

4.64.1 Detailed Description

This is the class for a line to do some geometric manipulation.

Note this the theoretical line, ie it goes infinitely, if you want what most people think of as a line (ie with endpoints) use **ArLineSegment** (p. 226)

4.64.2 Member Function Documentation

4.64.2.1 bool ArLine::intersects (const ArLine * line, ArPose * pose) [inline]

finds the intersection of this line with another line.

Parameters:

line the line to check if it intersects with this line
pose if the lines intersect, the pose is set to the location

Returns:

true if they intersect, false if they do not

The documentation for this class was generated from the following file:

 \bullet ariaUtil.h

4.65 ArLineSegment Class Reference

This is the class for a line segment to do some geometric manipulation. #include <ariaUtil.h>

Public Methods

- ArLineSegment (double x1, double y1, double x2, double y2)

 Constructor with endpoints.
- virtual ~**ArLineSegment** ()

 Destructor.
- void **newEndPoints** (double x1, double y1, double x2, double y2)

 Gives the line some new end points (makes it a segment).
- bool intersects (const ArLine *line, ArPose *pose)

 Sees if a line intersects with this segment.
- bool intersects (ArLineSegment *line, ArPose *pose)

 Sees if a line segment intersects with this segment.
- bool **getPerpPoint** (**ArPose** pose, **ArPose** *perpPoint)

 Gets the point at which the given pose is perpindicular to the line.
- bool **getPerpPoint** (const **ArPose** *pose, **ArPose** *perpPoint)

 Gets the point at which the given pose is perpindicular to the line.
- const double getX1 (void) const
 Gets the x coordinate of the first endpoint (only use on segments).
- const double **getY1** (void) const

 Gets the y coordinate of the first endpoint (only use on segments).
- const double **getX2** (void) const

 Gets the x coordinate of the second endpoint (only use on segments).
- const double **getY2** (void) const

 Gets the y coordinate of the second endpoint (only use on segments).
- const double **getA** (void) const Gets the A line parameter.

- const double **getB** (void) const Gets the B line parameter.
- const double **getC** (void) const Gets the C line parameter.
- const bool linePointIsInSegment (ArPose *pose) const Internal function for seeing if a point on our line is within our segment.

Protected Attributes

- double **myX1**
 - $Internal\ function\ to\ set\ the\ parameters\ of\ the\ line\ from\ endpoitns.$
- double myY1

 Internal function to set the parameters of the line from endpoitns.
- double myX2

 Internal function to set the parameters of the line from endpoitns.
- double myY2

 Internal function to set the parameters of the line from endpoitns.

4.65.1 Detailed Description

This is the class for a line segment to do some geometric manipulation.

4.65.2 Member Function Documentation

4.65.2.1 bool ArLineSegment::getPerpPoint (const ArPose * pose, ArPose * perpPoint) [inline]

Gets the point at which the given pose is perpindicular to the line.

This is just a faster version for certain critical spots than the above one... use the above one if you can... If the point is beyond either segment end this will return false

Parameters:

pose the pointer of the pose to find the perp point of

pose the pointer of the pose to set to the found point

Returns:

true if the pose is within the bounds of the line segment

4.65.2.2 bool ArLineSegment::getPerpPoint (ArPose pose, ArPose * perpPoint) [inline]

Gets the point at which the given pose is perpindicular to the line.

If the point is beyond either segment end this will return false

Parameters:

pose the pointer of the pose to find the perp point of
pose the pointer of the pose to set to the found point

Returns

true if the pose is within the bounds of the line segment

4.65.2.3 bool ArLineSegment::intersects (ArLineSegment * line, ArPose * pose) [inline]

Sees if a line segment intersects with this segment.

Parameters:

line the line segment to check if it intersects with this line
pose if the lines intersect, the pose is set to the location

Returns:

true if they intersect, false if they do not

4.65.2.4 bool ArLineSegment::intersects (const ArLine * line, ArPose * pose) [inline]

Sees if a line intersects with this segment.

Parameters:

line the line to check if it intersects with this line
pose if the lines intersect, the pose is set to the location

Returns:

true if they intersect, false if they do not

The documentation for this class was generated from the following file:

 $\bullet \ \, {\rm ariaUtil.h}$

4.66 ArListPos Class Reference

has enum for position in list.

#include <ariaTypedefs.h>

Public Types

• enum Pos { FIRST = 1, LAST = 2 }

4.66.1 Detailed Description

has enum for position in list.

4.66.2 Member Enumeration Documentation

4.66.2.1 enum ArListPos::Pos

Enumeration values:

FIRST place item first in the list.

LAST place item last in the list.

The documentation for this class was generated from the following file:

• ariaTypedefs.h

4.67 ArLog Class Reference

Logging utility class.
#include <ArLog.h>

Public Types

- enum LogType { StdOut, StdErr, File, Colbert, None }
- enum LogLevel { Terse, Normal, Verbose }

Static Public Methods

- void log (LogLevel level, char *str,...)

 Log a message.
- void logPlain (LogLevel level, char *str)

 Log a message without varargs (wrapper for java) Log a message.
- bool init (LogType type, LogLevel level, const char *fileName="")

 Initialize the logging utility.
- void **close** ()

 Close the logging utility.

4.67.1 Detailed Description

Logging utility class.

ArLog is a utility class to log all messages from **Aria** (p. 205) to a choosen destintation. Messages can be logged to stdout, stderr, a file, and turned off completely. Logging by default is set to stdout. The level of logging can be changed as well. Allowed levels are Terse, Normal, and Verbose. By default the level is set to Normal.

4.67.2 Member Enumeration Documentation

4.67.2.1 enum ArLog::LogLevel

Enumeration values:

Terse Use terse logging.

Normal Use normal logging.Verbose Use verbose logging.

4.67.2.2 enum ArLog::LogType

Enumeration values:

StdOut Use stdout for logging.

StdErr Use stderr for logging.

File Use a file for logging.

Colbert Use a Colbert stream for logging.

None Disable logging.

4.67.3 Member Function Documentation

4.67.3.1 bool ArLog::init (LogType type, LogLevel level, const char * fileName = "") [static]

Initialize the logging utility.

Initialize the logging utility by supplying the type of logging and the level of logging. If the type is File, the fileName needs to be supplied.

Parameters:

```
type type of Logginglevel level of loggingfileName the name of the file for File type of logging
```

4.67.3.2 void ArLog::log (LogLevel level, char * str, ...) [static]

Log a message.

This function is used like printf(). If the supplied level is less than or equal to the set level, it will be printed.

Parameters:

```
level level of logging
str printf() like formating string
```

The documentation for this class was generated from the following files:

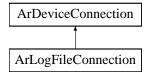
- ArLog.h
- ArLog.cpp

4.68 ArLogFileConnection Class Reference

For connecting through a log file.

#include <ArLogFileConnection.h>

Inheritance diagram for ArLogFileConnection::



Public Types

enum Open { OPEN_FILE_NOT_FOUND = 1, OPEN_NOT_A_-LOG_FILE }

Public Methods

• ArLogFileConnection ()

Constructor.

• virtual ~ArLogFileConnection ()

Destructor also closes connection.

• int open (const char *fname=NULL)

Opens a connection to the given host and port.

• virtual bool **openSimple** (void)

Opens the connection again, using the values from setLocation or.

• virtual int **getStatus** (void)

Gets the status of the connection, which is one of the enum status.

• virtual bool close (void)

Closes the connection.

 virtual int read (const char *data, unsigned int size, unsigned int ms-Wait=0)

Reads data from connection.

- virtual int write (const char *data, unsigned int size)

 Writes data to connection.
- virtual const char * **getOpenMessage** (int messageNumber)

 Gets the string of the message associated with opening the device.
- virtual **ArTime getTimeRead** (int index)

 Gets the time data was read in.
- virtual bool isTimeStamping (void)
 sees if timestamping is really going on or not.
- const char * **getLogFile** (void)

 Gets the name of the host connected to.
- int internalOpen (void)

 Internal function used by open and openSimple.

4.68.1 Detailed Description

For connecting through a log file.

4.68.2 Member Enumeration Documentation

4.68.2.1 enum ArLogFileConnection::Open

Enumeration values:

OPEN_FILE_NOT_FOUND Can't find the file.
OPEN_NOT_A_LOG_FILE Doesn't look like a log file.

4.68.3 Member Function Documentation

4.68.3.1 bool ArLogFileConnection::close (void) [virtual]

Closes the connection.

Returns:

whether the close succeeded or not

Reimplemented from ArDeviceConnection (p. 126).

4.68.3.2 const char * ArLogFileConnection::getLogFile (void)

Gets the name of the host connected to.

Returns:

the name of the log file

4.68.3.3 const char * ArLogFileConnection::getOpenMessage (int messageNumber) [virtual]

Gets the string of the message associated with opening the device.

Each class inherited from this one has an open method which returns 0 for success or an integer which can be passed into this function to obtain a string describing the reason for failure

Parameters:

messageNumber the number returned from the open

Returns:

the error description associated with the messageNumber

Reimplemented from ArDeviceConnection (p. 126).

4.68.3.4 int ArLogFileConnection::getStatus (void) [virtual]

Gets the status of the connection, which is one of the enum status.

Gets the status of the connection, which is one of the enum status. If you want to get a string to go along with the number, use getStatusMessage

Returns:

the status of the connection

See also:

getStatusMessage (p. 127)

Reimplemented from ArDeviceConnection (p. 126).

4.68.3.5 ArTime ArLogFileConnection::getTimeRead (int index) [virtual]

Gets the time data was read in.

Parameters:

index looks like this is the index back in the number of bytes last read in

Returns:

the time the last read data was read in

Reimplemented from **ArDeviceConnection** (p. 127).

4.68.3.6 bool ArLogFileConnection::isTimeStamping (void) [virtual]

sees if timestamping is really going on or not.

Returns:

true if real timestamping is happening, false otherwise

Reimplemented from **ArDeviceConnection** (p. 127).

4.68.3.7 int ArLogFileConnection::open (const char * fname = NULL)

Opens a connection to the given host and port.

Parameters:

fname the file to connect to, if NULL (default) then robot.log

Returns:

0 for success, otherwise one of the open enums

See also:

getOpenMessage (p. 235)

4.68.3.8 int ArLogFileConnection::read (const char * data, unsigned int size, unsigned int msWait = 0) [virtual]

Reads data from connection.

Reads data from connection

Parameters:

data pointer to a character array to read the data into size maximum number of bytes to read msWait read blocks for this many milliseconds (not at all for < 0)

Returns:

number of bytes read, or -1 for failure

See also:

```
write (p. 237), writePacket (p. 129)
```

Reimplemented from ArDeviceConnection (p. 128).

4.68.3.9 int ArLogFileConnection::write (const char * data, unsigned int size) [virtual]

Writes data to connection.

Writes data to connection

Parameters:

data pointer to a character array to write the data from
size number of bytes to write

Returns:

number of bytes read, or -1 for failure

See also:

```
read (p. 236), writePacket (p. 129)
```

Reimplemented from ArDeviceConnection (p. 128).

The documentation for this class was generated from the following files:

- ArLogFileConnection.h
- ArLogFileConnection.cpp

4.69 ArMath Class Reference

This class has static members to do common math operations.

#include <ariaUtil.h>

Static Public Methods

- double addAngle (double ang1, double ang2)

 This adds two angles together and fixes the result to [-180, 180].
- double **subAngle** (double ang1, double ang2)

 This subtracts one angle from another and fixes the result to [-180,180].
- double fixAngle (double angle)

 Takes an angle and returns the angle in range (-180,180).
- double degToRad (double deg)

 Converts an angle in degrees to an angle in radians.
- double radToDeg (double rad)

 Converts an angle in radians to an angle in degrees.
- double **cos** (double angle)

 Finds the cos, from angles in degrees.
- double **sin** (double angle)

 Finds the sin, from angles in degrees.
- double **atan2** (double y, double x)

 Finds the arctan of the given y/x pair.
- bool **angleBetween** (double angle, double startAngle, double endAngle) Finds if one angle is between two other angles.
- double **fabs** (double val)

 Finds the absolute value of a double.
- int roundInt (double val)

 Finds the closest integer to double given.
- void **pointRotate** (double *x, double *y, double th)

 Rotates a point around 0 by degrees given.

• long random (void)

Returns a long between 0 and some arbitrary huge number.

- double **distanceBetween** (double x1, double y1, double x2, double y2) Finds the distance between two coordinates.
- double **squaredDistanceBetween** (double x1, double y1, double x2, double y2)

Finds the squared distance between two coordinates.

4.69.1 Detailed Description

This class has static members to do common math operations.

4.69.2 Member Function Documentation

4.69.2.1 double ArMath::addAngle (double ang1, double ang2) [inline, static]

This adds two angles together and fixes the result to [-180, 180].

Parameters:

```
ang1 first angleang2 second angle, added to first
```

Returns:

```
sum of the angles, in range [-180,180]
```

See also:

```
subAngle (p. 242), fixAngle (p. 241)
```

4.69.2.2 double ArMath::atan2 (double y, double x) [inline, static]

Finds the arctan of the given y/x pair.

Parameters:

```
\boldsymbol{y} the y distance
```

 \boldsymbol{x} the x distance

Returns:

the angle y and x form

4.69.2.3 double ArMath::cos (double angle) [inline, static]

Finds the cos, from angles in degrees.

Parameters:

angle angle to find the cos of, in degrees

Returns:

the cos of the angle

See also:

sin (p. 242)

4.69.2.4 double ArMath::degToRad (double deg) [inline, static]

Converts an angle in degrees to an angle in radians.

Parameters:

deg the angle in degrees

Returns:

the angle in radians

See also:

radToDeg (p. 241)

4.69.2.5 double ArMath::distanceBetween (double x1, double y1, double x2, double y2) [inline, static]

Finds the distance between two coordinates.

${\bf Parameters:}$

```
x1 the first coords x position
```

y1 the first coords y position

x2 the second coords x position

y2 the second coords y position

Returns:

the distance between (x1, y1) and (x2, y2)

4.69.2.6 double ArMath::fabs (double val) [inline, static]

Finds the absolute value of a double.

Parameters:

val the number to find the absolute value of

Returns:

the absolute value of the number

4.69.2.7 double ArMath::fixAngle (double angle) [inline, static]

Takes an angle and returns the angle in range (-180,180].

Parameters:

angle the angle to fix

Returns:

the angle in range (-180,180]

See also:

addAngle (p. 239), subAngle (p. 242)

4.69.2.8 double ArMath::radToDeg (double rad) [inline, static]

Converts an angle in radians to an angle in degrees.

Parameters:

rad the angle in radians

Returns:

the angle in degrees

See also:

degToRad (p. 240)

4.69.2.9 int ArMath::roundInt (double val) [inline, static]

Finds the closest integer to double given.

Parameters:

val the double to find the nearest integer to

Returns

the integer the value is nearest to

4.69.2.10 double ArMath::sin (double angle) [inline, static]

Finds the sin, from angles in degrees.

Parameters:

angle angle to find the sin of, in degrees

Returns:

the sin of the angle

See also:

cos (p. 240)

4.69.2.11 double ArMath::squaredDistanceBetween (double x1, double y1, double x2, double y2) [inline, static]

Finds the squared distance between two coordinates.

use this only where speed really matters

Parameters:

x1 the first coords x position

y1 the first coords y position

x2 the second coords x position

y2 the second coords y position

Returns:

the distance between (x1, y1) and (x2, y2)

4.69.2.12 double ArMath::subAngle (double ang1, double ang2) [inline, static]

This subtracts one angle from another and fixes the result to [-180,180].

Parameters:

ang1 first angle

ang2 second angle, subtracted from first angle

Returns:

resulting angle, in range [-180,180]

See also:

```
\mathbf{addAngle}\; (\mathbf{p}.\, 239) , \mathbf{fixAngle}\; (\mathbf{p}.\, 241)
```

The documentation for this class was generated from the following file:

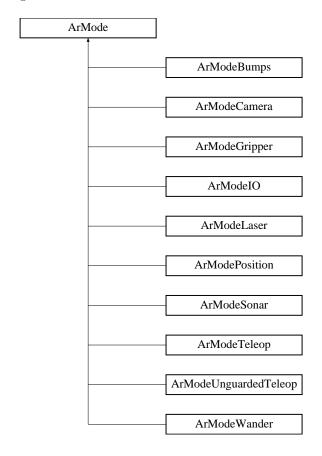
 \bullet ariaUtil.h

4.70 ArMode Class Reference

A class for different modes, mostly as related to keyboard input.

#include <ArMode.h>

Inheritance diagram for ArMode::



Public Methods

- **ArMode** (**ArRobot** *robot, const char *name, char key, char key2) *Constructor.*
- virtual ~**ArMode** ()

 Destructor.
- const char * **getName** (void)

Gets the name of the mode.

• virtual void activate (void)=0

The function called when the mode is activated, subclass must provide.

• virtual void deactivate (void)=0

The function called when the mode is deactivated, subclass must provide.

• virtual void userTask (void)

The ArMode's user task, don't need one, subclass must provide if needed.

• virtual void **help** (void)

The mode's help print out... subclass must provide if needed.

• bool baseActivate (void)

The base activation, it MUST be called by inheriting classes, and inheriting classes MUST return if this returns false.

• bool baseDeactivate (void)

The base deactivation, it MUST be called by inheriting classes, and inheriting classes MUST return if this returns false.

• char **getKey** (void)

An internal function to get the first key this is bound to.

• char **getKey2** (void)

An internal function to get the second key this is bound to.

Static Public Methods

• void **baseHelp** (void)

This is the base help function, its internal, bound to ? and h and H.

4.70.1 Detailed Description

A class for different modes, mostly as related to keyboard input.

Each mode is going to need to add its keys to the keyHandler... each mode should only use the keys 1-0, the arrow keys (movement), the space bar (stop), z (zoom in), x (zoom out), and e (exercise)... then when its activate is called by that key handler it needs to first deactivate the ourActiveMode (if its not

itself, in which case its done) then add its key handling stuff... activate and deactivate will need to add and remove their user tasks (or call the base class activate/deactivate to do it) as well as the key handling things for their other part of modes. This mode will ALWAYS bind help to /, ?, h, and H when the first instance of an ArMode is made.

4.70.2 Constructor & Destructor Documentation

4.70.2.1 ArMode::ArMode (ArRobot * robot, const char * name, char key, char key2)

Constructor.

Parameters:

robot the robot we're attaching to

name the name of this mode

key the first key to switch to this mode on... it can be '\0' if you don't want to use this

 \pmb{key} the first key to switch to this mode on... it can be '\0' if you don't want to use this

4.70.3 Member Function Documentation

4.70.3.1 bool ArMode::baseActivate (void)

The base activation, it MUST be called by inheriting classes, and inheriting classes MUST return if this returns false.

Inheriting modes must first call this to get their user task called and to deactivate the active mode.... if it returns false then the inheriting class must return, as it means that his mode is already active

4.70.3.2 bool ArMode::baseDeactivate (void)

The base deactivation, it MUST be called by inheriting classes, and inheriting classes MUST return if this returns false.

This gets called when the mode is deactivated, it removes the user task from the robot

4.70.3.3 virtual void ArMode::help (void) [inline, virtual]

The mode's help print out... subclass must provide if needed.

This is called as soon as a mode is activated, and should give directions on to what keys do what and what this mode will do

Reimplemented in **ArModeTeleop** (p. 254), **ArModeUnguardedTeleop** (p. 256), **ArModeWander** (p. 258), **ArModeGripper** (p. 251), **ArModeCamera** (p. 249), and **ArModeSonar** (p. 253).

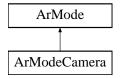
- \bullet ArMode.h
- \bullet ArMode.cpp

4.71 ArModeCamera Class Reference

Mode for controlling the gripper.

#include <ArModes.h>

Inheritance diagram for ArModeCamera::



Public Methods

• ArModeCamera (ArRobot *robot, const char *name, char key, char key2)

Constructor.

• virtual ~ArModeCamera ()

Destructor.

• virtual void activate (void)

 $The \ function \ called \ when \ the \ mode \ is \ activated, \ subclass \ must \ provide.$

• virtual void **deactivate** (void)

The function called when the mode is deactivated, subclass must provide.

• virtual void userTask (void)

The ArMode (p. 244)'s user task, don't need one, subclass must provide if needed.

• virtual void **help** (void)

The mode's help print out... subclass must provide if needed.

4.71.1 Detailed Description

Mode for controlling the gripper.

4.71.2 Member Function Documentation

4.71.2.1 void ArModeCamera::help (void) [virtual]

The mode's help print out... subclass must provide if needed.

This is called as soon as a mode is activated, and should give directions on to what keys do what and what this mode will do

Reimplemented from **ArMode** (p. 246).

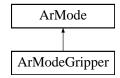
- \bullet ArModes.h
- ArModes.cpp

4.72 ArModeGripper Class Reference

Mode for controlling the gripper.

#include <ArModes.h>

Inheritance diagram for ArModeGripper::



Public Methods

• ArModeGripper (ArRobot *robot, const char *name, char key, char key2)

Constructor.

• virtual ~**ArModeGripper** ()

Destructor.

• virtual void activate (void)

 $The \ function \ called \ when \ the \ mode \ is \ activated, \ subclass \ must \ provide.$

• virtual void **deactivate** (void)

The function called when the mode is deactivated, subclass must provide.

• virtual void **userTask** (void)

The ArMode (p. 244)'s user task, don't need one, subclass must provide if needed.

• virtual void **help** (void)

The mode's help print out... subclass must provide if needed.

4.72.1 Detailed Description

Mode for controlling the gripper.

4.72.2 Member Function Documentation

4.72.2.1 void ArModeGripper::help (void) [virtual]

The mode's help print out... subclass must provide if needed.

This is called as soon as a mode is activated, and should give directions on to what keys do what and what this mode will do

Reimplemented from **ArMode** (p. 246).

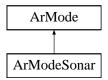
- \bullet ArModes.h
- \bullet ArModes.cpp

4.73 ArModeSonar Class Reference

Mode for displaying the sonar.

#include <ArModes.h>

Inheritance diagram for ArModeSonar::



Public Methods

• ArModeSonar (ArRobot *robot, const char *name, char key, char key2)

Constructor.

• virtual ~ArModeSonar ()

Destructor.

• virtual void activate (void)

 $The \ function \ called \ when \ the \ mode \ is \ activated, \ subclass \ must \ provide.$

• virtual void **deactivate** (void)

The function called when the mode is deactivated, subclass must provide.

• virtual void **userTask** (void)

The ArMode (p. 244)'s user task, don't need one, subclass must provide if needed.

• virtual void **help** (void)

The mode's help print out... subclass must provide if needed.

4.73.1 Detailed Description

Mode for displaying the sonar.

4.73.2 Member Function Documentation

4.73.2.1 void ArModeSonar::help (void) [virtual]

The mode's help print out... subclass must provide if needed.

This is called as soon as a mode is activated, and should give directions on to what keys do what and what this mode will do

Reimplemented from **ArMode** (p. 246).

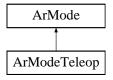
- \bullet ArModes.h
- ArModes.cpp

4.74 ArModeTeleop Class Reference

Mode for teleoping the robot with joystick + keyboard.

#include <ArModes.h>

Inheritance diagram for ArModeTeleop::



Public Methods

• ArModeTeleop (ArRobot *robot, const char *name, char key, char key2)

Constructor.

• virtual ~ArModeTeleop ()

Destructor.

• virtual void activate (void)

The function called when the mode is activated, subclass must provide.

• virtual void deactivate (void)

The function called when the mode is deactivated, subclass must provide.

• virtual void **help** (void)

The mode's help print out... subclass must provide if needed.

4.74.1 Detailed Description

Mode for teleoping the robot with joystick + keyboard.

4.74.2 Member Function Documentation

4.74.2.1 void ArModeTeleop::help (void) [virtual]

The mode's help print out... subclass must provide if needed.

This is called as soon as a mode is activated, and should give directions on to what keys do what and what this mode will do

Reimplemented from **ArMode** (p. 246).

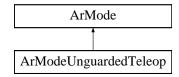
- \bullet ArModes.h
- ArModes.cpp

4.75 ArModeUnguardedTeleop Class Reference

Mode for teleoping the robot with joystick + keyboard.

#include <ArModes.h>

Inheritance diagram for ArModeUnguardedTeleop::



Public Methods

• ArModeUnguardedTeleop (ArRobot *robot, const char *name, char key, char key2)

Constructor.

• virtual ~ArModeUnguardedTeleop ()

Destructor.

• virtual void activate (void)

The function called when the mode is activated, subclass must provide.

• virtual void deactivate (void)

The function called when the mode is deactivated, subclass must provide.

• virtual void **help** (void)

The mode's help print out... subclass must provide if needed.

4.75.1 Detailed Description

Mode for teleoping the robot with joystick + keyboard.

4.75.2 Member Function Documentation

4.75.2.1 void ArModeUnguardedTeleop::help (void) [virtual]

The mode's help print out... subclass must provide if needed.

This is called as soon as a mode is activated, and should give directions on to what keys do what and what this mode will do

Reimplemented from **ArMode** (p. 246).

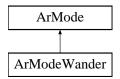
- \bullet ArModes.h
- ArModes.cpp

4.76 ArModeWander Class Reference

Mode for wandering around.

#include <ArModes.h>

Inheritance diagram for ArModeWander::



Public Methods

• ArModeWander (ArRobot *robot, const char *name, char key, char key2)

Constructor.

• virtual ~ArModeWander ()

Destructor.

• virtual void activate (void)

The function called when the mode is activated, subclass must provide.

• virtual void deactivate (void)

The function called when the mode is deactivated, subclass must provide.

• virtual void **help** (void)

The mode's help print out... subclass must provide if needed.

4.76.1 Detailed Description

Mode for wandering around.

4.76.2 Member Function Documentation

4.76.2.1 void ArModeWander::help (void) [virtual]

The mode's help print out... subclass must provide if needed.

This is called as soon as a mode is activated, and should give directions on to what keys do what and what this mode will do

Reimplemented from **ArMode** (p. 246).

- \bullet ArModes.h
- ArModes.cpp

4.77 ArModule Class Reference

Dynamicly loaded module base class, read warning in more.

#include <ArModule.h>

Public Methods

• ArModule ()

Constructor.

• virtual ~**ArModule** ()

Destructor.

- virtual bool init (ArRobot *robot, void *argument=NULL)=0

 Initialize the module. The module should use the supplied ArRobot (p. 355)
 pointer.
- virtual bool exit ()=0

 Close down the module and have it exit.
- ArRobot * getRobot ()

 Get the ArRobot (p. 355) pointer the module should be using.
- void **setRobot** (**ArRobot** *robot)

 Set the **ArRobot** (p. 355) pointer.

Protected Attributes

• ArRobot * myRobot

Stored ArRobot (p. 355) pointer that the module should use.

4.77.1 Detailed Description

Dynamicly loaded module base class, read warning in more.

Right now only one module's init will be called, that is the first one, its a bug that I just don't have time to fix at the moment. I'll get to it when I have time or if someone needs it... someone else wrote this code so it'll take me a little longer to fix it.

This class defines a dyanicmly loaded module of code. This is usefull for an application to load piece of code that it does not know about. The ArModule defines and interface in which to invoke that piece of code that the program does not know about. For instance, a module could contain an **ArAction** (p. 39) and the modules **init**() (p. 261) could instantiate the **ArAction** (p. 39) and add it to the supplied **ArRobot** (p. 355). The **init**() (p. 261) takes a reference to an **ArRobot** (p. 355). The module should use that robot for its purposes. If the module wants to use more robots, assuming there are multiple robots, it can use **Aria::getRobotList**() (p. 206) to find all the **ArRobot** (p. 355) instantiated. The module should do all its clean up in **exit**() (p. 260).

The user should derive their own class from ArModule and implement the **init**() (p. 261) and **exit**() (p. 260) functions. The users code should always clean up when **exit**() (p. 260) is called. **exit**() (p. 260) is called right before the module (dynamic library .dll/.so) is closed and removed from the program.

The macro ARDEF_MODULE() must be called within the .cpp file of the users module. A global instance of the users module must be defined and a reference to that instance must be passed to ARDEF_MODULE(). This allows the **Ar-ModuleLoader** (p. 263) to find the users module class and invoke it.

One thing to note about the use of code wrapped in ArModules and staticly linking in that code. To be able to staticly link .cpp files which contain an ArModule, the define of ARIA_STATIC should be defined. This will cause the ARDEF_MODULE() to do nothing. If it defined its normal functions and variables, the linker would fail to staticly link in multiple modules since they all have symbols with the same name.

See also **ArModuleLoader** (p. 263) to see how to load an ArModule into a program.

See also the example programs simpleMod.cpp and simpleModule.cpp. For a more complete example, see the example programs joydriveActionMod.cpp and joydriveActionModule.cpp.

4.77.2 Member Function Documentation

4.77.2.1 virtual bool ArModule::init (ArRobot * robot, void * argument = NULL) [pure virtual]

Initialize the module. The module should use the supplied **ArRobot** (p. 355) pointer.

Parameters:

robot Robot this module should attach to, can be NULL for none, so make sure you handle that case

modArgument an optional string argument to the module, this defaults

to NULL, you'll need to cast this to whatever you want it to be... you'll want to document this clearly with the module

- \bullet ArModule.h
- \bullet ArModule.cpp

4.78 ArModuleLoader Class Reference

Dynamic **ArModule** (p. 260) loader. #include <ArModuleLoader.h>

Public Types

enum Status { STATUS_SUCCESS = 0, STATUS_ALREADY_LOADED, STATUS_FAILED_OPEN, STATUS_INVALID, STATUS_INIT_FAILED, STATUS_EXIT_FAILED, STATUS_NOT_FOUND }

Static Public Methods

• Status load (const char *modName, ArRobot *robot, void *modArgument=NULL, bool quiet=false)

Load an ArModule (p. 260).

• Status reload (const char *modName, ArRobot *robot, void *mod-Argument=NULL, bool quiet=false)

Close and then reload an ArModule (p. 260).

- Status close (const char *modName, bool quiet=false)
 - Close an ArModule (p. 260).
- void closeAll ()

Close all open ArModule (p. 260).

4.78.1 Detailed Description

Dynamic **ArModule** (p. 260) loader.

The ArModuleLoader is used to load ArModules into a program and invoke them.

See also **ArModule** (p. 260) to see how to define an **ArModule** (p. 260).

See also the example programs simpleMod.cpp and simpleModule.cpp. For a more complete example, see the example programs joydriveActionMod.cpp and joydriveActionModule.cpp.

4.78.2 Member Enumeration Documentation

4.78.2.1 enum ArModuleLoader::Status

Enumeration values:

STATUS_SUCCESS Load succeded.

STATUS_ALREADY_LOADED Module already loaded.

STATUS_FAILED_OPEN Could not find or open the module.

STATUS_INVALID Invalid module file format.

STATUS_INIT_FAILED The module failed its init stage.

STATUS_EXIT_FAILED The module failed its exit stage.

STATUS_NOT_FOUND The module was not found.

4.78.3 Member Function Documentation

4.78.3.1 ArModuleLoader::Status ArModuleLoader::close (const char * modName, bool quiet = false) [static]

Close an **ArModule** (p. 260).

Calls **ArModule::exit()** (p. 260) on the module, then closes the library.

Parameters:

modName filename of the module without the extension (.dll or .so) quiet whether to print out a message if this fails or not, defaults to false

4.78.3.2 ArModuleLoader::Status ArModuleLoader::load (const char * modName, ArRobot * robot, void * modArgument = NULL, bool quiet = false) [static]

Load an **ArModule** (p. 260).

THIS ONLY LOADS one init on the module right now, if its called again it'll load the same init over. I'll fix it later... read the more verbose description in ArModule.h.

Takes a string name of the module which is just the file name of the module without the extension (.dll or .so). It will figure out the correct extension based on wheter its a Linux or Windows build. It will also uses the standard operating systems ability to find the library. So the library must be located within the PATH variable for Windows and the LD_LIBRARY_PATH for Linux. You can also just give the absolute path to the library, or the relative path from the directory the program was started in (ie ./simpleMod). The **ArModule**

(p. 260) will be passed the **ArRobot** (p. 355) reference that **load**() (p. 264) takes. This is the **ArRobot** (p. 355) that the **ArModule** (p. 260) will use for its processing.

Parameters:

modName filename of the module without the extension (.dll or .so)

 ${f robot}$ ArRobot (p. 355) reference which the module is to use, this can be NULL

modArgument A void pointer argument to pass to the module, if its a const value you'll need to cast it to a non-const value to get it to work (for example if you were using a constant string). This value defaults to NULL.

quiet whether to print out a message if this fails or not, defaults to false

4.78.3.3 ArModuleLoader::Status ArModuleLoader::reload (const char * modName, ArRobot * robot, void * modArgument = NULL, bool quiet = false) [static]

Close and then reload an **ArModule** (p. 260).

reload() (p. 265) is similiar to load() (p. 264), except that it will call close() (p. 264) on the module and then call load() (p. 264).

Parameters:

modName filename of the module without the extension (.dll or .so)

- ArModuleLoader.h
- ArModuleLoader.cpp

4.79 ArMutex Class Reference

Mutex wrapper class.

#include <ArMutex.h>

Public Types

• enum Status { STATUS_FAILED_INIT = 1, STATUS_FAILED, STATUS_ALREADY_LOCKED }

Public Methods

• ArMutex ()

Constructor.

 \bullet virtual ${\sim}\mathbf{ArMutex}$ ()

Destructor.

• virtual int lock ()

Lock the mutex.

• virtual int **tryLock** ()

Try to lock the mutex, but do not block.

• virtual int unlock ()

Unlock the mutex, allowing another thread to obtain the lock.

- virtual const char * **getError** (int messageNumber) const Get a human readable error message from an error code.
- virtual MutexType & **getMutex** ()

 Get a reference to the underlying mutex variable.

4.79.1 Detailed Description

Mutex wrapper class.

This class wraps the operating systems mutex functions. It allows mutually exclusive access to a critical section. This is extremely usefull for multiple threads which want to use the same variable. ArMutex simply uses the POSIX pthread interface in an object oriented manner. It also applies the same concept to Windows using Windows own abilities to restrict access to critical sections.

4.79.2 Member Enumeration Documentation

4.79.2.1 enum ArMutex::Status

Enumeration values:

STATUS_FAILED_INIT Failed to initialize.

STATUS_FAILED General failure.

STATUS_ALREADY_LOCKED Mutex already locked.

4.79.3 Member Function Documentation

4.79.3.1 int ArMutex::lock () [virtual]

Lock the mutex.

Lock the mutex. This function will block until no other thread has this mutex locked. If it returns 0, then it obtained the lock and the thread is free to use the critical section that this mutex protects. Else it returns an error code. See **getError**() (p. 266).

4.79.3.2 int ArMutex::tryLock () [virtual]

Try to lock the mutex, but do not block.

Try to lock the mutex. This function will not block if another thread has the mutex locked. It will return instantly if that is the case. It will return STATUS_-ALREADY_LOCKED if another thread has the mutex locked. If it obtains the lock, it will return 0.

- ArMutex.h
- ArMutex_LIN.cpp
- ArMutex_WIN.cpp

4.80 ArNetServer Class Reference

Class for running a simple net server to send/recv commands via text.

#include <ArNetServer.h>

Public Methods

• ArNetServer ()

Constructor.

• ~ArNetServer ()

Destructor.

• bool **open** (**ArRobot** *robot, unsigned int port, const char *password, bool multipleClients)

Initializes the server.

• void **close** (void)

Closes the server.

bool addCommand (const char *command, ArFunctor3< char **, int,
 ArSocket *> *functor, const char *help)

Adds a new command.

• bool remCommand (const char *command)

Removes a command.

• void **sendToAllClients** (const char *str,...)

Sends the given string to all the clients.

• void sendToAllClientsPlain (const char *str)

 $Sends\ the\ given\ string\ to\ all\ the\ clients,\ no\ varargs,\ wrapper\ for\ java.$

• bool isOpen (void)

Sees if the server is running and open.

• void runOnce (void)

the internal sync task we use for our loop.

• void internalGreeting (ArSocket *socket)

the internal function that gives the greeting message.

• void internalHelp (ArSocket *socket)

The internal function that does the help.

- void **internalHelp** (char **argv, int argc, **ArSocket** *socket)

 The internal function for the help cb.
- void **internalEcho** (char **argv, int argc, **ArSocket** *socket)

 The internal function for echo.
- void internalQuit (char **argv, int argc, ArSocket *socket)

 The internal function for closing this connection.
- void **internalShutdown** (char **argv, int argc, **ArSocket** *socket)

 The internal function for shutting down.

4.80.1 Detailed Description

Class for running a simple net server to send/recv commands via text.

This class is for running a simple net server which will have a list of commands to use and a fairly simple set of interactions... Start the server with the open function, add commands with the addCommand function and remove commands with remCommand, and close the server with the close function.

4.80.2 Member Function Documentation

4.80.2.1 bool ArNetServer::addCommand (const char * command, ArFunctor3< char **, int, ArSocket *> * functor, const char * help)

Adds a new command.

This adds a command to the list, when the command is given the broken up argv and argc are given along with the socket it came from (so that acks can occur)

4.80.2.2 bool ArNetServer::open (ArRobot * robot, unsigned int port, const char * password, bool multipleClients)

Initializes the server.

Open the server, if you supply a robot this will run in the robots attached, if you do not supply a robot then it will be open and you'll have to call runOnce yourself (this is only recommended for advanced users)

Parameters:

robot the robot that this should be attached to and run in the sync task of or NULL not to run in any robot's task

port the port to start up the service on

password the password needed to use the service

multiple Clients if false only one client is allowed to connect, if false multiple clients are allowed to connect or just one

Returns:

true if the server could be started, false otherwise

4.80.2.3 bool ArNetServer::remCommand (const char * command)

Removes a command.

Parameters:

command the command to remove

Returns:

true if the command was there to remove, false otherwise

4.80.2.4 void ArNetServer::sendToAllClients (const char * str, ...)

Sends the given string to all the clients.

This sends the given string to all the clients, this string cannot be more than 2048 number of bytes

- ArNetServer.h
- ArNetServer.cpp

4.81 ArNetServerConnection Class Reference

this class holds the information related to specific connections.

#include <ArNetServer.h>

Public Methods

• ArNetServerConnection ()

Constructor.

• ~ArNetServerConnection ()

Destructor.

• ArSocket * getSocket (void)

Gets the socket it uses.

• char * readString (void)

Reads in a string if there are any to read.

• void **setEcho** (bool on)

Sets if we're echoing or not.

• bool **getEcho** (void)

Gets if we're echoing or not.

• void **doEcho** (void)

Do the echoing (if needed).

4.81.1 Detailed Description

this class holds the information related to specific connections.

4.81.2 Member Function Documentation

4.81.2.1 char * ArNetServerConnection::readString (void)

Reads in a string if there are any to read.

This reads a single string (terminated with

Returns:

if we can't read from the socket anymore we return NULL if we can read but there's no command we return a string thats of 0 length... otherwise we return a string

- ArNetServer.h
- $\bullet \ \, ArNetServer.cpp$

4.82 ArP2Arm Class Reference

Arm Control class.

#include <ArP2Arm.h>

Public Types

 enum State { SUCCESS, ALREADY_INITED, NOT_INITED, ROBOT_NOT_SETUP, NO_ARM_FOUND, COMM_FAILED, COULD_NOT_OPEN_PORT, COULD_NOT_SET_UP_PORT, ALREADY_CONNECTED, NOT_CONNECTED, INVALID_JOINT, INVALID_POSITION }

General error conditions possible from most of the arm related functions.

• enum PacketType { StatusPacket, InfoPacket }

Type of arm packet identifiers. Used in ArP2Arm::setPacketCB() (p. 275).

• enum StatusType { StatusOff = 0, StatusSingle = 1, Status-Continuous = 2 }

Type of status packets to request for. Used in ArP2Arm::requestStatus() (p. 283).

Public Methods

• ArP2Arm ()

Constructor.

• virtual ~ArP2Arm ()

Destructor.

• void **setRobot** (**ArRobot** *robot)

Set the robot to use to talk to the arm.

• virtual **State init** ()

Init the arm class.

• virtual **State uninit** ()

Uninit the arm class.

• virtual **State powerOn** (bool doWait=true)

Power on the arm.

• virtual **State powerOff** ()

Power off the arm.

• virtual State requestInfo ()

Request the arm info packet.

• virtual State requestStatus (StatusType status)

Request the arm status packet.

• virtual State requestInit ()

Request arm initialization.

• virtual **State checkArm** (bool waitForResponse=true)

Check to see if the arm is still connected.

• virtual **State home** (int joint=-1)

Home the arm.

• virtual **State park** ()

Home the arm and power if off.

• virtual **State moveTo** (int joint, float pos, unsigned char vel=0)

Move a joint to a position in degrees.

• virtual **State moveToTicks** (int joint, unsigned char pos)

Move a joint to a position in low level arm controller ticks.

• virtual **State moveStep** (int joint, float pos, unsigned char vel=0)

Move a joint step degrees.

• virtual State moveStepTicks (int joint, signed char pos)

Move a joint step ticks.

• virtual **State moveVel** (int joint, int vel)

Set the joint to move at the given velocity.

• virtual **State stop** ()

Stop the arm.

• virtual **State setAutoParkTimer** (int waitSecs)

Set the auto park timer value.

• virtual State setGripperParkTimer (int waitSecs)

Set the gripper park timer value.

• virtual void **setStoppedCB** (**ArFunctor** *func)

Set the arm stopped callback.

• virtual void setPacketCB (ArFunctor1 < PacketType > *func)

set the arm packet callback.

• virtual std::string **getArmVersion** ()

Get the arm version.

• virtual float **getJointPos** (int joint)

Get the joints position in degrees.

• virtual unsigned char **getJointPosTicks** (int joint)

Get the joints position in ticks.

• virtual bool **getMoving** (int joint=-1)

Check to see if the arm is moving.

• virtual bool isPowered ()

Check to see if the arm is powered.

• virtual bool isGood ()

Check to see if the arm is communicating.

• virtual int getStatus ()

Get the two byts of status info from P2OS.

• virtual ArTime getLastStatusTime ()

Get when the last arm status packet came in.

• virtual **ArRobot** * **getRobot** ()

Get the robot that the arm is on.

virtual P2ArmJoint * getJoint (int joint)

Get the joints data structure.

 virtual bool convertDegToTicks (int joint, float pos, unsigned char *ticks) Converts degrees to low level arm controller ticks.

• virtual bool **convertTicksToDeg** (int joint, unsigned char pos, float *degrees)

Converts low level arm controller ticks to degrees.

Static Public Attributes

- const int **ArmJoint1** = 0x1

 Bit for joint 1 in arm status byte.
- const int **ArmJoint2** = 0x2

 Bit for joint 2 in arm status byte.
- const int **ArmJoint3** = 0x4

 Bit for joint 3 in arm status byte.
- const int **ArmJoint4** = 0x8

 Bit for joint 4 in arm status byte.
- const int **ArmJoint5** = 0x10

 Bit for joint 5 in arm status byte.
- const int **ArmJoint6** = 0x20

 Bit for joint 6 in arm status byte.
- const int **ArmGood** = 0x100

 Bit for arm good state in arm status byte.
- const int **ArmInited** = 0x200

 Bit for arm initialized in arm status byte.
- const int **ArmPower** = 0x400

 Bit for arm powered on in arm status byte.
- const int **ArmHoming** = 0x800 Bit for arm homing in arm status byte.
- int NumJoints = 6

 Number of joints that the arm has.

4.82.1 Detailed Description

Arm Control class.

ArP2Arm is the interface to the AROS/P2OS-based Pioneer 2 Arm servers. The P2 Arm is attached to the robot's microcontroller via an auxiliary serial port.

To use ArmP2, you must first set up an **ArRobot** (p. 355) and have it connect with the robot. The **ArRobot** (p. 355) needs to be run so that it reads and writes packets to and from server. The easiest way is **ArRobot::runAsync()** (p. 397) which runs the **ArRobot** (p. 355) in its own thread.

Then call **ArP2Arm::setRobot**() (p. 273) with **ArRobot** (p. 355), and finally initialized with ArmP2::init(). Once initialized, use the various ArP2Arm methods to power the P2 Arm servos, move joints, and so on.

For simple examples on how to use ArP2Arm, look in the **Aria** (p. 205)/examples directory for P2ArmSimple.cpp and P2ArmJoydrive.cpp.

See the Aria (p. 205) documentation on how to use Aria (p. 205).

4.82.2 Member Enumeration Documentation

4.82.2.1 enum ArP2Arm::PacketType

Type of arm packet identifiers. Used in ArP2Arm::setPacketCB() (p. 275).

Enumeration values:

StatusPacket The status packet type.

InfoPacket The info packet type.

4.82.2.2 enum ArP2Arm::State

General error conditions possible from most of the arm related functions.

Enumeration values:

SUCCESS Succeded.

ALREADY_INITED The class is already initialized.

NOT_INITED The class is not initialized.

ROBOT_NOT_SETUP The **ArRobot** (p. 355) class is not setup properly.

NO_ARM_FOUND The arm can not be found.

COMM_FAILED Communications has failed.

COULD_NOT_OPEN_PORT Could not open the communications port.

COULD_NOT_SET_UP_PORT Could not setup the communications port.

ALREADY_CONNECTED Already connected to the arm.

NOT_CONNECTED Not connected with the arm, connect first.

INVALID_JOINT Invalid joint specified.

INVALID_POSITION Invalid position specified.

4.82.2.3 enum ArP2Arm::StatusType

Type of status packets to request for. Used in **ArP2Arm::requestStatus**() (p. 283).

Enumeration values:

StatusOff Stop sending status packets.

StatusSingle Send a single status packets.

StatusContinuous Send continuous packets. Once every 100ms.

4.82.3 Member Function Documentation

4.82.3.1 ArP2Arm::State ArP2Arm::checkArm (bool waitForResponse = true) [virtual]

Check to see if the arm is still connected.

Requests that P2OS checks to see if the arm is still alive and immediately exits. This is not a full init and differs that P2OS will still accept arm commands and the arm will not be parked. If P2OS fails to find the arm it will change the status byte accordingly and stop accepting arm related commands except for init commands. If the parameter waitForResponse is true then **checkArm**() (p. 278) will wait the appropriate amoutn of time and check the status of the arm. If you wish to do the waiting else where the arm check sequence takes about 200ms, so the user should wait 300ms then send a **ArP2Arm::request-Status**() (p. 283) to get the results of the check arm request. Since there is a very noticable time delay, the user should use the **ArP2Arm::setPacketCB**() (p. 275) to set a callback so the user knows when the packet has been recieved.

This can be useful for telling if the arm is still alive. The arm controller can be powered on/off seperately from the robot.

Parameters:

waitForResponse cause the function to block until their is a response

See also:

```
requestInit (p. 282), setPacketCB (p. 275)
```

4.82.3.2 ArP2Arm::State ArP2Arm::home (int joint = -1) [virtual]

Home the arm.

Tells the arm to go to the home position. While the arm is homing, the status byte will reflect it with the **ArP2Arm::ArmHoming** (p. 276) flag. If joint is set to -1, then all the joints will be homed at a safe speed. If a single joint is specified, that joint will be told to go to its home position at the current speed its set at.

Parameters:

joint home only that joint

4.82.3.3 ArP2Arm::State ArP2Arm::init (void) [virtual]

Init the arm class.

Initialize the P2 Arm class. This must be called before anything else. The **setRobot**() (p. 273) must be called to let ArP2Arm know what instance of an **ArRobot** (p. 355) to use. It talks to the robot and makes sure that there is an arm on it and it is in a good condition. The AROS/P2OS arm servers take care of AUX port serial communications with the P2 Arm controller.

4.82.3.4 ArP2Arm::State ArP2Arm::moveStep (int joint, float pos, unsigned char vel = 0) [virtual]

Move a joint step degrees.

Step the joint post degrees from its current position at the given speed. If vel is 0, then the currently set speed will be used.

See ArP2Arm::moveToTicks() (p. 281) for a description of how positions are defined. See ArP2Arm::moveVel() (p. 281) for a description of how speeds are defined.

Parameters:

```
joint the joint to movepos the position in degrees to stepvel the speed at which to move. 0 will use the currently set speed
```

See also:

```
moveTo (p. 280), moveVel (p. 281)
```

4.82.3.5 ArP2Arm::State ArP2Arm::moveStepTicks (int *joint*, signed char *pos*) [virtual]

Move a joint step ticks.

Move the joint posticks from its current position. A tick is the arbitrary position value that the arm controller uses. The arm controller uses a single unsigned byte to represent all the possible positions in the range of the servo for each joint. So the range of ticks is 0-255 which is mapped to the physical range of the servo. Due to the design of the arm, certain joints range are limited by the arm itself. P2OS will bound the position to physical range of each joint. This is a lower level of controlling the arm position than using **ArP2Arm::move-To()** (p. 280). **ArP2Arm::moveStep()** (p. 279) uses a conversion factor which converts degrees to ticks.

Parameters:

```
joint the joint to move
pos the position, in ticks, to move to
```

See also:

moveStep (p. 279)

4.82.3.6 ArP2Arm::State ArP2Arm::moveTo (int joint, float pos, unsigned char vel = 0) [virtual]

Move a joint to a position in degrees.

Move the joint to the position at the given speed. If vel is 0, then the currently set speed will be used. The position is in degrees. Each joint has about a +-90 degree range, but they all differ due to the design.

See ArP2Arm::moveToTicks() (p. 281) for a description of how positions are defined. See ArP2Arm::moveVel() (p. 281) for a description of how speeds are defined.

Parameters:

```
joint the joint to move
pos the position in degrees to move to
vel the speed at which to move. 0 will use the currently set speed
```

See also:

```
moveToTicks (p. 281), moveVel (p. 281)
```

4.82.3.7 ArP2Arm::State ArP2Arm::moveToTicks (int *joint*, unsigned char *pos*) [virtual]

Move a joint to a position in low level arm controller ticks.

Move the joint to the given position in ticks. A tick is the arbitrary position value that the arm controller uses. The arm controller uses a single unsigned byte to represent all the possible positions in the range of the servo for each joint. So the range of ticks is 0-255 which is mapped to the physical range of the servo. Due to the design of the arm, certain joints range are limited by the arm itself. P2OS will bound the position to physical range of each joint. This is a lower level of controlling the arm position than using **ArP2Arm::move-To()** (p. 280). **ArP2Arm::move-To()** (p. 280) uses a conversion factor which converts degrees to ticks.

Parameters:

```
joint the joint to move
pos the position, in ticks, to move to
```

See also:

moveTo (p. 280)

4.82.3.8 ArP2Arm::State ArP2Arm::moveVel (int *joint*, int *vel*) [virtual]

Set the joint to move at the given velocity.

Set the joints velocity. The arm controller has no way of controlling the speed of the servos in the arm. So to control the speed of the arm, P2OS will incrementaly send a string of position commands to the arm controller to get the joint to move to its destination. To vary the speed, the amount of time to wait between each point in the path is varied. The velocity parameter is simply the number of milliseconds to wait between each point in the path. 0 is the fastest and 255 is the slowest. A reasonable range is around 10-40.

Parameters:

```
joint the joint to move
vel the velocity to move at
```

4.82.3.9 ArP2Arm::State ArP2Arm::powerOff () [virtual]

Power off the arm.

Powers off the arm. This should only be called when the arm is in a good position to power off. Due to the design, it will go limp when the power is turned off.

A more safe way to power off the arm is to use the **ArP2Arm::park**() (p. 274) function. Which will home the arm, then power if off.

See also:

park (p. 274)

4.82.3.10 ArP2Arm::State ArP2Arm::powerOn (bool doSleep = true) [virtual]

Power on the arm.

Powers on the arm. The arm will shake for up to 2 seconds after powering on. If the arm is told to move before it stops shaking, that vibration can be amplified by moving. The default is to wait the 2 seconds for the arm to settle down.

Parameters:

doSleep if true, sleeps 2 seconds to wait for the arm to stop shaking

4.82.3.11 ArP2Arm::State ArP2Arm::requestInfo () [virtual]

Request the arm info packet.

Requests the arm info packet from P2OS and immediately returns. This packet will be sent during the next 100ms cycle of P2OS. Since there is a very noticable time delay, the user should use the **ArP2Arm::setPacketCB()** (p. 275) to set a callback so the user knows when the packet has been recieved.

See also:

setPacketCB (p. 275)

4.82.3.12 ArP2Arm::State ArP2Arm::requestInit () [virtual]

Request arm initialization.

Requests that P2OS initialize the arm and immediately returns. The arm initialization procedure takes about 700ms to complete and a little more time for the status information to be relayed back to the client. Since there is a very noticable time delay, the user should use the **ArP2Arm::setPacketCB**() (p. 275) to set a callback so the user knows when the arm info packet has been recieved. Then wait about 800ms, and send a **ArP2Arm::requestStatus**() (p. 283) to get the results of the init request. While the init is proceding, P2OS will ignore all arm related commands except requests for arm status and arm info packets.

ArP2Arm::checkArm() (p. 278) can be used to periodicly check to make sure that the arm controller is still alive and responding.

See also:

checkArm (p. 278), setPacketCB (p. 275)

4.82.3.13 ArP2Arm::State ArP2Arm::requestStatus (StatusType status) [virtual]

Request the arm status packet.

Requests the arm status packet from P2OS and immediately returns. This packet will be sent during the next 100ms cycle of P2OS. Since there is a very noticable time delay, the user should use the **ArP2Arm::setPacket-CB()** (p. 275) to set a callback so the user knows when the packet has been recieved.

See also:

setPacketCB (p. 275)

4.82.3.14 ArP2Arm::State ArP2Arm::setAutoParkTimer (int waitSecs) [virtual]

Set the auto park timer value.

P2OS will automaticly park the arm if it gets no arm related packets after wait-Secs. This is to help protect the arm when the program looses connection with P2OS. Set the value to 0 to disable this timer. Default wait is 10 minutes.

Parameters:

waitSecs seconds to wait till parking the arm when idle

4.82.3.15 ArP2Arm::State ArP2Arm::setGripperParkTimer (int waitSecs) [virtual]

Set the gripper park timer value.

P2OS/AROS automatically park the gripper after its been closed for more than waitSecs. The gripper servo can overheat and burnout if it is holding something for more than 10 minutes. Care must be taken to ensure that this does not happen. If you wish to manage the gripper yourself, you can disable this timer by setting it to 0.

Parameters:

waitSecs seconds to wait till parking the gripper once it has begun to grip something

4.82.3.16 ArP2Arm::State ArP2Arm::stop () [virtual]

Stop the arm.

Stop the arm from moving. This overrides all other actions except for the arms initilization sequence.

4.82.3.17 ArP2Arm::State ArP2Arm::uninit () [virtual]

Uninit the arm class.

Uninitialize the arm class. This simply asks the arm to park itself and cleans up its internal state. To completely uninitialize the P2 Arm itself have the **ArRobot** (p. 355) disconnect from P2OS.

The documentation for this class was generated from the following files:

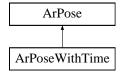
- ArP2Arm.h
- ArP2Arm.cpp

4.83 ArPose Class Reference

The class which represents a position.

#include <ariaUtil.h>

Inheritance diagram for ArPose::



Public Methods

- **ArPose** (double x=0, double y=0, double th=0) Constructor, with optional initial values.
- **ArPose** (const ArPose &pose)

 Copy Constructor.
- virtual \sim **ArPose** () Destructor.
- virtual void **setPose** (double x, double y, double th=0)

 Sets the position to the given values.
- virtual void **setPose** (ArPose position)

 Sets the position equal to the given position.
- void **setX** (double x)

 Sets the x position.
- void **setY** (double y)

 Sets the y position.
- void **setTh** (double th)

 Sets the heading.
- void **setThRad** (double th)

 Sets the heading, using radians.

- double **getX** (void) const Gets the x position.
- double **getY** (void) const Gets the y position.
- double **getTh** (void) const Gets the heading.
- double **getThRad** (void) const Gets the heading, in radians.
- void **getPose** (double *x, double *y, double *th=NULL) const Gets the whole position in one function call.
- virtual double **findDistanceTo** (ArPose position) const Finds the distance from this position to the given position.
- virtual double **squaredFindDistanceTo** (ArPose position) const Finds the square distance from this position to the given position.
- virtual double **findAngleTo** (ArPose position) const Finds the angle between this position and the given position.
- virtual void log (void) const

 Logs the coordinates using ArLog (p. 231).

4.83.1 Detailed Description

The class which represents a position.

This class represents a robot position with heading. The heading defaults to 0, and so does not need to be used (this avoids having 2 types of positions). Everything in the class is inline so it should be fast.

4.83.2 Constructor & Destructor Documentation

4.83.2.1 ArPose::ArPose (double x = 0, double y = 0, double th = 0) [inline]

Constructor, with optional initial values.

Sets the position with the given values, can be used with no variables, with just x and y, or with x, y, and th

Parameters:

```
\boldsymbol{x} the position to set the x position to, default of 0
```

y the position to set the y position to, default of 0

th the position to set the th position to, default of 0

4.83.3 Member Function Documentation

4.83.3.1 virtual double ArPose::findAngleTo (ArPose position) const [inline, virtual]

Finds the angle between this position and the given position.

Parameters:

position the position to find the angle to

Returns:

the angle to the given position from this instance, in degrees

4.83.3.2 virtual double ArPose::findDistanceTo (ArPose position) const [inline, virtual]

Finds the distance from this position to the given position.

Parameters:

position the position to find the distance to

Returns:

the distance to the position from this instance

4.83.3.3 void ArPose::getPose (double * x, double * y, double * th = NULL) const [inline]

Gets the whole position in one function call.

Gets the whole position at once, by giving it 2 or 3 pointers to doubles. If you give the function a null pointer for a value it won't try to use the null pointer, so you can pass in a NULL if you don't care about that value. Also note that the defaults to NULL so you can use this with just x and y.

Parameters:

 \boldsymbol{x} a pointer to a double to set the x position to

y a pointer to a double to set the y position to

th a pointer to a double to set the heading to, defaults to NULL

4.83.3.4 virtual void ArPose::setPose (ArPose position) [inline, virtual]

Sets the position equal to the given position.

Parameters:

position the position value this instance should be set to

4.83.3.5 virtual void ArPose::setPose (double x, double y, double th = 0) [inline, virtual]

Sets the position to the given values.

Sets the position with the given three values, but the theta does not need to be given as it defaults to 0.

Parameters:

 \boldsymbol{x} the position to set the x position to

y the position to set the v position to

th the position to set the th position to, default of 0

4.83.3.6 virtual double ArPose::squaredFindDistanceTo (ArPose position) const [inline, virtual]

Finds the square distance from this position to the given position.

This is only here for speed, if you aren't doing this thousands of times a second don't use this one use findDistanceTo

Parameters:

position the position to find the distance to

Returns:

the distance to the position from this instance

The documentation for this class was generated from the following file:

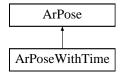
• ariaUtil.h

4.84 ArPoseWithTime Class Reference

A subclass of pose that also has the time the pose was taken.

#include <ariaUtil.h>

Inheritance diagram for ArPoseWithTime::



4.84.1 Detailed Description

A subclass of pose that also has the time the pose was taken.

The documentation for this class was generated from the following file:

 \bullet ariaUtil.h

4.85 ArPref Class Reference

Preference instance. Used by ArPreferences.

#include <ArPref.h>

Public Types

• enum ValType { Integer, Double, Boolean, String }

Public Methods

• **ArPref** (int section, int pref, const char *name, const char *val, const char *validVals, **ValType** valType, const char *comment)

Constructor.

• ArPref (const ArPref &pref)

Copy constructor.

• virtual ~**ArPref** ()

Destructor.

• virtual bool **getBool** ()

Get the value as a boolean.

• virtual int **getInt** ()

Get the value as an integer.

• virtual double **getDouble** ()

Get the value as a double.

• virtual std::string **getString** ()

Get the value as a std::string.

• virtual bool **setBool** (bool val, bool append=false)

Set the value to be the supplied boolean.

• virtual bool **setInt** (int val, bool append=false)

Set the value to be the supplied integer.

• virtual bool **setDouble** (double val, bool append=false)

Set the value to be the supplied doule.

- virtual bool **setString** (const char *val, bool append=false)

 Set the value to be the supplied std::string.
- virtual int **getSetCount** ()

Get the number of values that would be in the set regardless of type.

- virtual int **getBoolSet** (bool *boolArray, int size)

 Get the value as multiple booleans.
- virtual int **getIntSet** (int *intArray, int size)

 Get the value as multiple integers.
- virtual int **getDoubleSet** (double *doubleArray, int size)

 Get the value as multiple doubles.
- virtual int **getStringSet** (std::string *stringArray, int size)

 Get the value as multiple std::strings.
- virtual bool **setBoolSet** (bool append, int count,...)

 Set the value to be the supplied booleans.
- virtual bool **setIntSet** (bool append, int count,...)

 Set the value to be the supplied integers.
- virtual bool **setDoubleSet** (bool append, int count,...)

 Set the value to be the supplied doubles.

4.85.1 Detailed Description

Preference instance. Used by ArPreferences.

This represents an individual preference which is loaded from compiled in defaults of from a preferences file. A preference can be one of four different types: Integer, Double, Boolean, String. The preference itself is stored as a string. There are accessors which convert from string to the desired format and vice versa: **getBool**() (p. 292), **getInt**() (p. 293), **getDouble**() (p. 293), **getString**() (p. 294), **setBool**() (p. 294), **setInt**() (p. 295), **setDouble**() (p. 295), **setString**() (p. 296). A preference can also have a set of values of all the same type. In the file would look like:

<key> <int> <int> <int> ...

The 'set' accessors can deal with an abitrary amount of values: **getBool-Set**() (p. 292), **getIntSet**() (p. 293), **getDoubleSet**() (p. 293), **getStringSet**() (p. 294), **setBoolSet**() (p. 295), **setIntSet**() (p. 296), **setDoubleSet**() (p. 295).

A preference can have an array of valid values. When the file is loaded, Ar-Preferences checks all the values from the file against the supplied valid values. The check is done with a string compare. It is most usefull for string values. So it will apply to numbers as long as they are formated in the correct way.

4.85.2 Member Enumeration Documentation

4.85.2.1 enum ArPref::ValType

Enumeration values:

Integer integer number value.

Double double number value.

Boolean boolean value, expressed as 'true' or 'false' in the file.

String a string value.

4.85.3 Member Function Documentation

4.85.3.1 bool ArPref::getBool (void) [virtual]

Get the value as a boolean.

Get the value, formating it correctly. If the preference is not of the boolean type or not found, it will return false.

4.85.3.2 int ArPref::getBoolSet (bool * boolArray, int size) [virtual]

Get the value as multiple booleans.

Get the value, formating it correctly. If the preference is not of the boolean type or not found, it will return false.

Parameters:

boolArray the array to fill out with the values
size the size of the passed in array

Returns:

the number of values put into the array

4.85.3.3 double ArPref::getDouble (void) [virtual]

Get the value as a double.

Get the value, formating it correctly. If the preference is not of the double type or not found, it will return 0.0.

4.85.3.4 int ArPref::getDoubleSet (double * doubleArray, int size) [virtual]

Get the value as multiple doubles.

Get the value, formating it correctly. If the preference is not of the double type or not found, it will return an empty list.

Parameters:

doubleArray the array to fill out with the values
size the size of the passed in array

Returns:

the number of values put into the array

4.85.3.5 int ArPref::getInt (void) [virtual]

Get the value as an integer.

Get the value, formating it correctly. If the preference is not of the integer type or not found, it will return 0.

4.85.3.6 int ArPref::getIntSet (int * intArray, int size) [virtual]

Get the value as multiple integers.

Get the value, formating it correctly. If the preference is not of the integer type or not found, it will return an empty list.

Parameters:

intArray the array to fill out with the values
size the size of the passed in array

Returns:

the number of values put into the array

4.85.3.7 int ArPref::getSetCount () [virtual]

Get the number of values that would be in the set regardless of type.

Get the number of values that is contained in this preference. This is independent of the type of values. Use this to figure out how big of an array that you need to get a set of values.

4.85.3.8 std::string ArPref::getString (void) [virtual]

Get the value as a std::string.

Get the value, formating it correctly. If the preference is not of the string type or not found, it will return "".

4.85.3.9 int ArPref::getStringSet (std::string * stringArray, int size) [virtual]

Get the value as multiple std::strings.

Get the value, formating it correctly. If the preference is not of the string type or not found, it will return an empty list.

Parameters:

stringArray the array to fill out with the values

size the size of the passed in array

Returns:

the number of values put into the array

4.85.3.10 bool ArPref::setBool (bool val, bool append = false) [virtual]

Set the value to be the supplied boolean.

If 'append' is true, a copy of this instance with the supplied value will be created and added to the ArPreferences.

Parameters:

 ${\it val}$ the value to set the preference to

append create a new instance of this preference

4.85.3.11 bool ArPref::setBoolSet (bool append, int count, ...) [virtual]

Set the value to be the supplied booleans.

If 'append' is true, a copy of this instance with the supplied value will be created and added to the ArPreferences.

Parameters:

append create a new instance of this preference
count the number of values in the parameter list

4.85.3.12 bool ArPref::setDouble (double val, bool append = false) [virtual]

Set the value to be the supplied doule.

If 'append' is true, a copy of this instance with the supplied value will be created and added to the ArPreferences.

Parameters:

val the value to set the preference toappend create a new instance of this preference

4.85.3.13 bool ArPref::setDoubleSet (bool append, int count, ...) [virtual]

Set the value to be the supplied doubles.

If 'append' is true, a copy of this instance with the supplied value will be created and added to the ArPreferences.

Parameters:

append create a new instance of this preference count the number of values in the parameter list

4.85.3.14 bool ArPref::setInt (int *val*, bool *append* = false) [virtual]

Set the value to be the supplied integer.

If 'append' is true, a copy of this instance with the supplied value will be created and added to the ArPreferences.

Parameters:

val the value to set the preference toappend create a new instance of this preference

4.85.3.15 bool ArPref::setIntSet (bool append, int count, ...) [virtual]

Set the value to be the supplied integers.

If 'append' is true, a copy of this instance with the supplied value will be created and added to the ArPreferences.

Parameters:

append create a new instance of this preference
count the number of values in the parameter list

4.85.3.16 bool ArPref::setString (const char * val, bool append = false) [virtual]

Set the value to be the supplied std::string.

If 'append' is true, a copy of this instance with the supplied value will be created and added to the ArPreferences.

Parameters:

val the value to set the preference toappend create a new instance of this preference

The documentation for this class was generated from the following files:

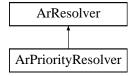
- ArPref.h
- ArPref.cpp

4.86 ArPriorityResolver Class Reference

(Default resolver), takes the action list and uses the priority to resolve.

#include <ArPriorityResolver.h>

Inheritance diagram for ArPriorityResolver::



Public Methods

• ArPriorityResolver ()

Constructor.

• virtual ~ArPriorityResolver ()

Destructor.

4.86.1 Detailed Description

(Default resolver), takes the action list and uses the priority to resolve.

This is the default resolver for **ArRobot** (p. 355), meaning if you don't do a non-normal init on the robot, or a setResolver, you'll have one these.

The documentation for this class was generated from the following files:

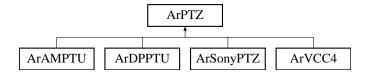
- ArPriorityResolver.h
- ArPriorityResolver.cpp

4.87 ArPTZ Class Reference

Base class which handles the PTZ cameras.

#include <ArPTZ.h>

Inheritance diagram for ArPTZ::



Public Methods

- ArPTZ (ArRobot *robot)
- virtual \sim **ArPTZ** ()

Destructor.

- virtual bool **init** (void)=0

 Initializes the camera.
- virtual bool **pan** (int degrees)=0

 Pans to the given degrees.
- virtual bool panRel (int degrees)=0
 Pans relative to current position by given degrees.
- virtual bool **tilt** (int degrees)=0

 Tilts to the given degrees.
- virtual bool **tiltRel** (int degrees)=0

 Tilts relative to the current position by given degrees.
- virtual bool **panTilt** (int degreesPan, int degreesTilt)=0

 Pans and tilts to the given degrees.
- virtual bool **panTiltRel** (int degreesPan, int degreesTilt)=0

 Pans and tilts relatives to the current position by the given degrees.
- virtual bool canZoom (void) const=0

 Returns true if camera can zoom (or rather, if it is controlled by this).

• virtual bool **zoom** (int zoomValue)

Zooms to the given value.

• virtual bool **zoomRel** (int zoomValue)

Zooms relative to the current value, by the given value.

• virtual int **getPan** (void) const=0

The angle the camera was last told to pan to.

• virtual int **getTilt** (void) const=0

The angle the camera was last told to tilt to.

• virtual int **getZoom** (void) const

The value the camera was last told to zoom to.

• virtual bool canGetRealPanTilt (void) const If this driver can tell the real pan/tilt angle.

• virtual int **getRealPan** (void) const The angle the camera says its at.

• virtual int **getRealTilt** (void) const

The angle the camera says its at.

virtual bool canGetRealZoom (void) const
 If this driver can tell the real zoom.

• virtual int **getRealZoom** (void) const The zoom the camera says its at.

• virtual int **getMaxPosPan** (void) const=0

Gets the highest positive degree the camera can pan to.

• virtual int **getMaxNegPan** (void) const=0

Gets the lowest negative degree the camera can pan to.

• virtual int **getMaxPosTilt** (void) const=0

Gets the highest positive degree the camera can tilt to.

• virtual int **getMaxNegTilt** (void) const=0

Gets the lowest negative degree the camera can tilt to.

• virtual int **getMaxZoom** (void) const

Gets the maximum value for the zoom on this camera.

• virtual int **getMinZoom** (void) const

Gets the lowest value for the zoom on this camera.

• virtual bool **setDeviceConnection** (**ArDeviceConnection** *connection, bool driveFromRobotLoop=true)

Sets the device connection to be used by this PTZ camera, if set this camera will send commands via this connection, otherwise its via robot.

• virtual ArDeviceConnection * getDeviceConnection (void)

Gets the device connection used by this PTZ camera.

• virtual bool **setAuxPort** (int auxPort)

Sets the aux port on the robot to be used to communicate with this device.

• virtual int **getAuxPort** (void)

Gets the port the device is set to communicate on.

virtual ArBasePacket * readPacket (void)

Reads a packet from the device connection, MUST NOT BLOCK.

virtual bool sendPacket (ArBasePacket *packet)

Sends a given packet to the camera (via robot or serial port, depending).

• virtual bool packetHandler (ArBasePacket *packet)

Handles a packet that was read from the device.

• virtual bool robotPacketHandler (ArRobotPacket *packet)

Handles a packet that was read by the robot.

• virtual void connectHandler (void)

Internal, attached to robot, inits the camera when robot connects.

• virtual void **sensorInterpHandler** (void)

Internal, for attaching to the robots sensor interp to read serial port.

4.87.1 Detailed Description

Base class which handles the PTZ cameras.

This class is mainly concerned with making all the cameras look the same for outgoing data, it is also set up to facilitate the acquisition of incoming data but that is described in the following paragraphs. There are two ways this can be used. The first is the simplest and default behavior and should be used by those whose cameras are attached to their robot's microcontroller, a **ArRobot** (p. 355) pointer is passed in to the contructor, this is where the commands will be sent to the robot via the robot's connection which will then send it along over the second serial port. The second way is to pass an **ArDeviceConnection** (p. 124) to setDeviceConnection, if this is done commands will be sent along the given serial port, this should ONLY be done if the camera is attached straight to a serial port on the computer this program is running on.

The next two paragraphs describe how to get data back from the cameras, but this base class is set up so that by default it won't try to get data back and assumes you're not trying to do that. If you are trying to get data back the important functions are packetHandler, robotPacketHandler and readPacket and you should read the docs on those.

If the camera is attached to the robot (and you are thus using the first method described in the first paragraph) then the only way to get data back is to send an **ArCommands::GETAUX** (p. 120), then set up a robotPacketHandler for the AUX id and have it call the packetHandler you set up in in the class.

If the camera is attached to the serial port on the computer (and thus the second method described in the first paragraph was used) then its more complicated... the default way is to just pass in an **ArDeviceConnection** (p. 124) to set-DeviceConnection and implement the readPacket method (which MUST not block), and every time through the robot loop readPacket (with the sensor-InterpHandler) will be called and any packets will be given to the packetHandler (which you need to implement in your class) to be processed. The other way to do this method is to pass both an ArDefaultConnection and false to setDevice-Connection, this means the camera will not be read at all by default, and you're on your own for reading the data in (ie like your own thread).

4.87.2 Constructor & Destructor Documentation

4.87.2.1 ArPTZ::ArPTZ (ArRobot * robot)

Parameters:

robot The robot this camera is attached to, can be NULL

4.87.3 Member Function Documentation

4.87.3.1 virtual bool ArPTZ::packetHandler (ArBasePacket * packet) [inline, virtual]

Handles a packet that was read from the device.

This should work for the robot packet handler or for packets read in from read-Packet (the joys of OO), but it can't deal with the need to check the id on robot packets, so you should check the id from robotPacketHandler and then call this one so that your stuff can be used by both robot and serial port connections.

Parameters:

packet the packet to handle

Returns:

true if this packet was handled (ie this knows what it is), false otherwise

Reimplemented in ArVCC4 (p. 510).

4.87.3.2 virtual ArBasePacket* ArPTZ::readPacket (void) [inline, virtual]

Reads a packet from the device connection, MUST NOT BLOCK.

This should read in a packet from the myConn connection and return a pointer to a packet if there was on to read in, or NULL if there wasn't one... this MUST not block if it is used with the default mode of being driven from the sensorInterpHandler, since that is on the robot loop.

Returns:

packet read in, or NULL if there was no packet read

Reimplemented in ArVCC4 (p. 511).

4.87.3.3 bool ArPTZ::robotPacketHandler (ArRobotPacket * packet) [virtual]

Handles a packet that was read by the robot.

This handles packets read in from the robot, this function should just check the ID of the robot packet and then return what packetHandler thinks of the packet.

Parameters:

packet the packet to handle

Returns:

true if the packet was handled (ie this konws what it is), false otherwise

4.87.3.4 bool ArPTZ::sendPacket (ArBasePacket * packet) [virtual]

Sends a given packet to the camera (via robot or serial port, depending).

Parameters:

packet the packet to send

Returns:

true if the packet could be sent, false otherwise

4.87.3.5 bool ArPTZ::setAuxPort (int auxPort) [virtual]

Sets the aux port on the robot to be used to communicate with this device.

Parameters:

auxPort The AUX port on the robot's microcontroller that the device is connected to. The C166 controller only has one port. The H8 has two.

Returns:

true if the port was valid (1 or 2). False otherwise.

4.87.3.6 bool ArPTZ::setDeviceConnection (ArDeviceConnection * connection, bool driveFromRobotLoop = true) [virtual]

Sets the device connection to be used by this PTZ camera, if set this camera will send commands via this connection, otherwise its via robot.

Parameters:

connection the device connection the camera is connected to, normally a serial port

driveFromRobotLoop if this is true then a sensor interp callback wil be set and that callback will read packets and call the packet handler on them

Returns:

true if the serial port is opened or can be opened, false otherwise

The documentation for this class was generated from the following files:

- \bullet ArPTZ.h
- \bullet ArPTZ.cpp

4.88 ArRangeBuffer Class Reference

This class is a buffer that holds ranging information.

#include <ArRangeBuffer.h>

Public Methods

• ArRangeBuffer (int size)

Constructor.

• virtual ~ArRangeBuffer ()

Destructor.

• size_t **getSize** (void) const

Gets the size of the buffer.

• void **setSize** (size_t size)

Sets the size of the buffer.

• ArPose getPoseTaken () const

Gets the pose of the robot when readings were taken.

• void setPoseTaken (ArPose p)

Sets the pose of the robot when readings were taken.

• void addReading (double x, double y)

Adds a new reading to the buffer.

• void beginInvalidationSweep (void)

Begins a walk through the getBuffer list of readings.

• void **invalidateReading** (std::list< **ArPoseWithTime** *>::iterator readingIt)

While doing an invalidation sweep a reading to the list to be invalidated.

• void endInvalidationSweep (void)

Ends the invalidation sweep.

• const std::list< $\mathbf{ArPoseWithTime} *> * \mathbf{getBuffer}$ (void) const

Gets a pointer to a list of readings.

• std::list< **ArPoseWithTime** *> * **getBuffer** (void)

Gets a pointer to a list of readings.

• double **getClosestPolar** (double startAngle, double endAngle, **ArPose** position, unsigned int maxRange, double *angle=NULL) const

Gets the closest reading, on a polar system.

• double **getClosestBox** (double x1, double y1, double x2, double y2, **Ar-Pose** position, unsigned int maxRange, **ArPose** *readingPos=NULL) const

Gets the closest reading, from a rectangular box, in robot LOCAL coords.

• void applyTransform (ArTransform trans)

Applies a transform to the buffer.

• void clear (void)

Clears all the readings in the range buffer.

• void clearOlderThan (int milliSeconds)

Resets the readings older than this many seconds.

• void clearOlderThanSeconds (int seconds)

Resets the readings older than this many seconds.

• void reset (void)

same as clear, but old name.

• void **beginRedoBuffer** (void)

This begins a redoing of the buffer.

• void **redoReading** (double x, double y)

Add a reading to the redoing of the buffer.

• void endRedoBuffer (void)

 $End\ redoing\ the\ buffer.$

4.88.1 Detailed Description

This class is a buffer that holds ranging information.

4.88.2 Constructor & Destructor Documentation

4.88.2.1 ArRangeBuffer::ArRangeBuffer (int size)

Constructor.

Parameters:

size The size of the buffer, in number of readings

4.88.3 Member Function Documentation

4.88.3.1 void ArRangeBuffer::addReading (double x, double y)

Adds a new reading to the buffer.

Parameters:

 \boldsymbol{x} the x position of the reading

 \boldsymbol{y} the y position of the reading

4.88.3.2 void ArRangeBuffer::applyTransform (ArTransform trans)

Applies a transform to the buffer.

Applies a transform to the buffers.. this is mostly useful for translating to/from local/global coords, but may have other uses

Parameters:

trans the transform to apply to the data

4.88.3.3 void ArRangeBuffer::beginInvalidationSweep (void)

Begins a walk through the getBuffer list of readings.

This is a set of funkiness used to invalid readings in the buffer. It is fairly complicated. But what you need to do, is set up the invalid sweeping with beginInvalidationSweep, then walk through the list of readings, and pass the iterator to a reading you want to invalidate to invalidateReading, then after you are all through walking the list call endInvalidationSweep. Look at the description of getBuffer for additional warnings.

See also:

invalidateReading (p. 310), endInvalidationSweep (p. 308)

4.88.3.4 void ArRangeBuffer::beginRedoBuffer (void)

This begins a redoing of the buffer.

To redo the buffer means that you're going to want to replace all of the readings in the buffer, and get rid of the ones that you don't replace (invalidate them). The three functions beginRedoBuffer, redoReading, and endRedoBuffer are all made to enable you to do this. What you do, is call **beginRedoBuffer**() (p. 308); then for each reading you want to be in the buffer, call **redoReading**(double x, double y) (p. 310), then when you are done, call **end-RedoBuffer**() (p. 308);

4.88.3.5 void ArRangeBuffer::endInvalidationSweep (void)

Ends the invalidation sweep.

See the description of beginInvalidationSweep, it describes how to use this function.

See also:

beginInvalidationSweep (p. 307), invalidateReading (p. 310)

4.88.3.6 void ArRangeBuffer::endRedoBuffer (void)

End redoing the buffer.

For a description of how to use this, see beginRedoBuffer

4.88.3.7 std::list< ArPoseWithTime *> * ArRangeBuffer::getBuffer (void)

Gets a pointer to a list of readings.

This function returns a pointer to a list that has all of the readings in it. This list is mostly for reference, ie for finding some particular value or for using the readings to draw them. Don't do any modification at all to the list unless you really know what you're doing... and if you do you'd better lock the rangeDevice this came from so nothing messes with the list while you are doing so.

Returns:

the list of positions this range buffer has

Gets a pointer to a list of readings.

This function returns a pointer to a list that has all of the readings in it. This list is mostly for reference, ie for finding some particular value or for using the readings to draw them. Don't do any modification at all to the list unless you really know what you're doing... and if you do you'd better lock the rangeDevice this came from so nothing messes with the list while you are doing so.

Returns:

the list of positions this range buffer has

4.88.3.9 double ArRangeBuffer::getClosestBox (double x1, double y1, double x2, double y2, ArPose startPos, unsigned int maxRange, ArPose * readingPos = NULL) const

Gets the closest reading, from a rectangular box, in robot LOCAL coords.

Gets the closest reading in a region defined by two points (opposeite points of a rectangle).

Parameters:

- x1 the x coordinate of one of the rectangle points
- y1 the y coordinate of one of the rectangle points
- x2 the x coordinate of the other rectangle point
- y2 the y coordinate of the other rectangle point
- **startPos** the position to find the closest reading to (usually the robots position)
- maxRange the maximum range to return (and what to return if nothing found)
- **readingPos** a pointer to a position in which to store the location of the closest position
- **position** the origin of the local coords for the definition of the coordinates, normally just ArRobot::getPosition

Returns:

if the return is >= 0 and $<= \max$ Range then this is the distance to the closest reading, if it is $>= \max$ Range, then there was no reading in the given section

4.88.3.10 double ArRangeBuffer::getClosestPolar (double startAngle, double endAngle, ArPose startPos, unsigned int maxRange, double * angle = NULL) const

Gets the closest reading, on a polar system.

Gets the closest reading in a region defined by startAngle going to endAngle... going counterclockwise (neg degrees to poseitive... with how the robot is set up, thats counterclockwise)... from -180 to 180... this means if you want the slice between 0 and 10 degrees, you must enter it as 0, 10, if you do 10, 0 you'll get the 350 degrees between 10 and 0... be especially careful with negative... for example -30 to -60 is everything from -30, around through 0, 90, and 180 back to -60... since -60 is actually to clockwise of -30

Parameters:

startAngle where to start the slice

endAngle where to end the slice, going clockwise from startAngle

startPos the position to find the closest reading to (usually the robots position)

maxRange the maximum range to return (and what to return if nothing found)

angle a pointer return of the angle to the found reading

position the origin of the local coords for the definition of the coordinates, normally just ArRobot::getPosition

Returns:

if the return is >=0 and <= maxRange then this is the distance to the closest reading, if it is >= maxRange, then there was no reading in the given section

4.88.3.11 void ArRangeBuffer::invalidateReading (std::list< ArPoseWithTime *>::iterator readingIt)

While doing an invalidation sweep a reading to the list to be invalidated.

See the description of beginInvalidationSweep, it describes how to use this function.

Parameters:

reading It the ITERATOR to the reading you want to get rid of

See also:

beginInvaladationSweep, endInvalidationSweep (p. 308)

4.88.3.12 void ArRangeBuffer::redoReading (double x, double y)

Add a reading to the redoing of the buffer.

For a description of how to use this, see beginRedoBuffer

Parameters:

- \boldsymbol{x} the x param of the coord to add to the buffer
- y the x param of the coord to add to the buffer

4.88.3.13 void ArRangeBuffer::setSize (size_t size)

Sets the size of the buffer.

If the new size is smaller than the current buffer it chops off the readings that are excess from the oldest readings... if the new size is larger then it just leaves room for the buffer to grow

Parameters:

 \boldsymbol{size} number of readings to set the buffer to

The documentation for this class was generated from the following files:

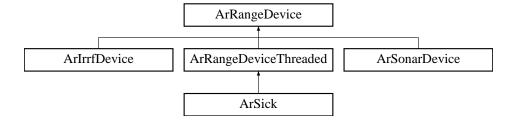
- ArRangeBuffer.h
- $\bullet \ \, ArRange Buffer.cpp$

4.89 ArRangeDevice Class Reference

The class for all devices which return range info (laser, sonar).

#include <ArRangeDevice.h>

Inheritance diagram for ArRangeDevice::



Public Methods

• ArRangeDevice (size_t currentBufferSize, size_t cumulativeBufferSize, const char *name, unsigned int maxRange)

Constructor.

• virtual ~ArRangeDevice ()

Destructor.

- virtual const char * **getName** (void) const Gets the name of the device.
- virtual void **setRobot** (**ArRobot** *robot)

 Sets the robot this device is attached to.
- virtual **ArRobot** * **getRobot** (void)

Gets the robot this device is attached to.

- virtual void **setCurrentBufferSize** (size_t size)

 Sets the size of the buffer for current readings.
- virtual void **setCumulativeBufferSize** (size_t size)

 Sets the size of the buffer for cumulative readings.
- virtual void **addReading** (double x, double y)

 Adds a reading to the buffer.

• virtual double **currentReadingPolar** (double startAngle, double end-Angle, double *angle=NULL) const

Gets the closest current reading in the given polar region.

• virtual double **cumulativeReadingPolar** (double startAngle, double endAngle, double *angle=NULL) const

Gets the closest cumulative reading in the given polar region.

• virtual double **currentReadingBox** (double x1, double y1, double x2, double y2, **ArPose** *readingPos=NULL) const

Gets the closest current reading from the given box region.

• virtual double **cumulativeReadingBox** (double x1, double y1, double x2, double y2, **ArPose** *readingPos=NULL) const

Gets the closest current reading from the given box region.

- virtual const **ArRangeBuffer** * **getCurrentRangeBuffer** (void) const Gets the current range buffer.
- ullet virtual const ${\bf ArRangeBuffer} * {\bf getCumulativeRangeBuffer}$ (void) const

Gets the cumulative range buffer.

 virtual const std::list< ArPoseWithTime *> * getCurrentBuffer (void) const

Gets the current buffer of readings.

 virtual const std::list< ArPoseWithTime *> * getCumulativeBuffer (void) const

Gets the current buffer of readings.

• virtual ArRangeBuffer * getCurrentRangeBuffer (void)

Gets the current range buffer.

virtual ArRangeBuffer * getCumulativeRangeBuffer (void)

Gets the cumulative range buffer.

• virtual std::list< **ArPoseWithTime** *> * **getCurrentBuffer** (void)

Gets the current buffer of readings.

virtual std::list< ArPoseWithTime *> * getCumulativeBuffer (void)

Gets the current buffer of readings.

• virtual const std::list< ArSensorReading *> * getRawReadings (void) const

Gets the raw unfiltered readings from the device.

• virtual void clearCurrentReadings (void)

Clears all the current readings.

• virtual void clearCumulativeReadings (void)

Clears all the cumulative readings.

• virtual void clearCumulativeOlderThan (int milliSeconds)

Clears all the cumulative readings older than this number of milliseconds.

• virtual void clearCumulativeOlderThanSeconds (int seconds)

Clears all the cumulative readings older than this number of seconds.

• virtual unsigned int **getMaxRange** (void)

Gets the maximum range for this device.

• virtual void **setMaxRange** (unsigned int maxRange)

Sets the maximum range for this device.

• virtual void **applyTransform** (**ArTransform** trans, bool do-Cumulative=true)

Applies a transform to the buffers.

• virtual int lockDevice ()

Lock this device.

• virtual int **tryLockDevice** ()

Try to lock this device.

• virtual int unlockDevice ()

Unlock this device.

4.89.1 Detailed Description

The class for all devices which return range info (laser, sonar).

This class has two buffers, a current buffer for storing just recent (relevant) readings, and a cumulative buffer for a longer history... the sizes of both can be set in the constructor.

This class should be used for all sensors like lasers and sonar, also note that it has the locking functions for such a time when there is a device like a laser that runs in its own thread, so that every device can be locked and unlocked and the users don't have to worry about the detail, because of functions on the **ArRobot** (p. 355) structure which check all of the ArRangeDevice's attached to a robot.

4.89.2 Constructor & Destructor Documentation

4.89.2.1 ArRangeDevice::ArRangeDevice (size_t currentBufferSize, size_t cumulativeBufferSize, const char * name, unsigned int maxRange)

Constructor.

Parameters:

current Buffer Size number of readings to store in the current buffer

cumulative Buffer Size number of readings to store in the cumulative buffer

name the name of this device

maxRange the max range of this device, if the device can't find a reading in a specified section, it returns this maxRange

4.89.3 Member Function Documentation

4.89.3.1 void ArRangeDevice::applyTransform (ArTransform trans, bool doCumulative = true) [virtual]

Applies a transform to the buffers.

Applies a transform to the buffers.. this is mostly useful for translating to/from local/global coords, but may have other uses

Parameters:

trans the transform to apply to the data

doCumulative whether to transform the cumulative buffer or not

4.89.3.2 double ArRangeDevice::cumulativeReadingBox (double x1, double y1, double x2, double y2, ArPose * pose = NULL) const [virtual]

Gets the closest current reading from the given box region.

Gets the closest reading in a region defined by two points (opposeite points of a rectangle) out of the cumulative buffer.

Parameters:

x1 the x coordinate of one of the rectangle points

y1 the y coordinate of one of the rectangle points

x2 the x coordinate of the other rectangle point

y2 the y coordinate of the other rectangle point

readingPos a pointer to a position in which to store the location of the closest position

Returns:

if the return is >=0 and <= maxRange then this is the distance to the closest reading, if it is >= maxRange, then there was no reading in the given section

4.89.3.3 double ArRangeDevice::cumulativeReadingPolar (double startAngle, double endAngle, double * angle = NULL) const [virtual]

Gets the closest cumulative reading in the given polar region.

Gets the closest reading in a region defined by startAngle going to endAngle... going counterclockwise (neg degrees to poseitive... with how the robot is set up, thats counterclockwise)... from -180 to 180... this means if you want the slice between 0 and 10 degrees, you must enter it as 0, 10, if you do 10, 0 you'll get the 350 degrees between 10 and 0... be especially careful with negative... for example -30 to -60 is everything from -30, around through 0, 90, and 180 back to -60... since -60 is actually to clockwise of -30

Parameters:

startAngle where to start the slice
endAngle where to end the slice, going clockwise from startAngle
position the position to find the closest reading to
angle a pointer return of the angle to the found reading

Returns:

if the return is >=0 and <= maxRange then this is the distance to the closest reading, if it is >= maxRange, then there was no reading in the given section

4.89.3.4 double ArRangeDevice::currentReadingBox (double x1, double y1, double x2, double y2, ArPose * pose = NULL) const [virtual]

Gets the closest current reading from the given box region.

Gets the closest reading in a region defined by two points (opposeite points of a rectangle) out of the current buffer.

Parameters:

x1 the x coordinate of one of the rectangle points

y1 the y coordinate of one of the rectangle points

x2 the x coordinate of the other rectangle point

y2 the y coordinate of the other rectangle point

readingPos a pointer to a position in which to store the location of the closest position

Returns:

if the return is >= 0 and $<= \max$ Range then this is the distance to the closest reading, if it is $>= \max$ Range, then there was no reading in the given section

4.89.3.5 double ArRangeDevice::currentReadingPolar (double startAngle, double endAngle, double * angle = NULL) const [virtual]

Gets the closest current reading in the given polar region.

Gets the closest reading in a region defined by startAngle going to endAngle... going counterclockwise (neg degrees to poseitive... with how the robot is set up, thats counterclockwise)... from -180 to 180... this means if you want the slice between 0 and 10 degrees, you must enter it as 0, 10, if you do 10, 0 you'll get the 350 degrees between 10 and 0... be especially careful with negative... for example -30 to -60 is everything from -30, around through 0, 90, and 180 back to -60... since -60 is actually to clockwise of -30

Parameters:

startAngle where to start the slice

endAngle where to end the slice, going clockwise from startAngle

position the position to find the closest reading to

angle a pointer return of the angle to the found reading

Returns:

if the return is >=0 and <= maxRange then this is the distance to the closest reading, if it is >= maxRange, then there was no reading in the given section

4.89.3.6 virtual const std::list<ArSensorReading *>* ArRangeDevice::getRawReadings (void) const [inline, virtual]

Gets the raw unfiltered readings from the device.

The raw readings are the full set of unfiltered readings from the device, they are the latest reading, you should manipulate the list you get from this function, the only manipulation of this list should be done by the range device itself. Its only pointers for speed.

4.89.3.7 virtual int ArRangeDevice::lockDevice (void) [inline, virtual]

Lock this device.

If you are also inheriting an ASyncTask you MUST override this to use the lock from the **ArASyncTask** (p. 110)

Reimplemented in ArRangeDeviceThreaded (p. 321).

4.89.3.8 void ArRangeDevice::setCumulativeBufferSize (size_t size) [virtual]

Sets the size of the buffer for cumulative readings.

If the new size is smaller than the current buffer it chops off the readings that are excess from the oldest readings... if the new size is larger then it just leaves room for the buffer to grow

Parameters:

size number of readings to set the buffer to

4.89.3.9 void ArRangeDevice::setCurrentBufferSize (size_t size) [virtual]

Sets the size of the buffer for current readings.

If the new size is smaller than the current buffer it chops off the readings that are excess from the oldest readings... if the new size is larger then it just leaves room for the buffer to grow

Parameters:

size number of readings to set the buffer to

4.89.3.10 virtual int ArRangeDevice::tryLockDevice (void) [inline, virtual]

Try to lock this device.

If you are also inheriting an ASyncTask you MUST override this to use the lock from the **ArASyncTask** (p. 110)

Reimplemented in ArRangeDeviceThreaded (p. 321).

4.89.3.11 virtual int ArRangeDevice::unlockDevice (void) [inline, virtual]

Unlock this device.

If you are also inheriting an ASyncTask you MUST override this to use the lock from the **ArASyncTask** (p. 110)

Reimplemented in ArRangeDeviceThreaded (p. 321).

The documentation for this class was generated from the following files:

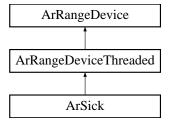
- ArRangeDevice.h
- ArRangeDevice.cpp

4.90 ArRangeDeviceThreaded Class Reference

A range device which can run in its own thread.

#include <ArRangeDeviceThreaded.h>

Inheritance diagram for ArRangeDeviceThreaded::



Public Methods

• ArRangeDeviceThreaded (size_t currentBufferSize, size_t cumulative-BufferSize, const char *name, unsigned int maxRange)

Constructor.

 $\bullet \ \, {\rm virtual} \sim \! \! \mathbf{ArRangeDeviceThreaded} \ () \\$

Destructor.

• virtual void * runThread (void *arg)=0

The functor you need to implement that will be the one executed by the thread.

• virtual void **run** (void)

 $Run\ in\ this\ thread.$

• virtual void **runAsync** (void)

Run in its own thread.

• virtual void **stopRunning** (void)

Stop the thread.

• virtual bool **getRunning** (void)

Get the running status of the thread.

• virtual bool **getRunningWithLock** (void)

Get the running status of the thread, locking around the variable.

• virtual int lockDevice (void)

Lock this device.

• virtual int tryLockDevice (void)

Try to lock this device.

• virtual int unlockDevice (void)

Unlock this device.

4.90.1 Detailed Description

A range device which can run in its own thread.

This is a range device thats threaded, it doesn't do multipleInheritance from both **ArASyncTask** (p. 110) and **ArRangeDevice** (p. 312) any more since JAVA doesn't support this and the wrapper software can't deal with it. Its still functionally the same however.

4.90.2 Member Function Documentation

4.90.2.1 virtual int ArRangeDeviceThreaded::lockDevice (void) [inline, virtual]

Lock this device.

If you are also inheriting an ASyncTask you MUST override this to use the lock from the **ArASyncTask** (p. 110)

Reimplemented from ArRangeDevice (p. 318).

4.90.2.2 virtual int ArRangeDeviceThreaded::tryLockDevice (void) [inline, virtual]

Try to lock this device.

If you are also inheriting an ASyncTask you MUST override this to use the lock from the **ArASyncTask** (p. 110)

Reimplemented from ArRangeDevice (p. 319).

4.90.2.3 virtual int ArRangeDeviceThreaded::unlockDevice (void) [inline, virtual]

Unlock this device.

If you are also inheriting an ASyncTask you MUST override this to use the lock from the ${\bf ArASyncTask}$ (p. 110)

Reimplemented from **ArRangeDevice** (p. 319).

The documentation for this class was generated from the following files:

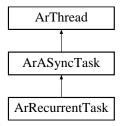
- $\bullet \ {\bf ArRange Device Threaded.h}$
- $\bullet \ {\bf ArRange Device Threaded.cpp}$

4.91 ArRecurrentTask Class Reference

Recurrent task (runs in its own thread).

#include <ArRecurrentTask.h>

Inheritance diagram for ArRecurrentTask::



Public Methods

• ArRecurrentTask ()

Constructor.

• \sim ArRecurrentTask ()

Descructor.

• virtual void **task** ()=0

The main run loop.

• void go ()

Starts up on cycle of the recurrent task.

• int done ()

Check if the task is running or not.

• void reset ()

Cancel the task and reset for the next cycle.

• void * runThread (void *ptr)

The main run loop.

4.91.1 Detailed Description

Recurrent task (runs in its own thread).

The ArRecurrentTask is a task that runs in its own thread. Recurrent tasks are asynchronous tasks that complete in a finite amount of time, and need to be reinvoked recurrently. A typical example is Saphira's localization task: it runs for a few hundred milliseconds, localizes the robot, and returns. Then the cycle starts over. The user simply needs to derive their own class from ArRecurrent-Task and define the $\mathbf{task}()$ (p. 324) function. This is the user code that will be called to execute the task body. Then, create an object of the class, and call the $\mathbf{go}()$ (p. 323) function to start the task. The status of the task can be checked with the $\mathbf{done}()$ (p. 324) function, which returns 0 if running, 1 if completed, and 2 if killed. $\mathbf{go}()$ (p. 323) can be called whenever the task is done to restart it. To stop the task in midstream, call $\mathbf{reset}()$ (p. 323). kill() kills off the thread, shouldn't be used unless exiting the async task permanently

4.91.2 Member Function Documentation

4.91.2.1 int ArRecurrentTask::done ()

Check if the task is running or not.

0 = running, 1 = finished normally, 2 = canceled

$\begin{array}{lll} \textbf{4.91.2.2} & \textbf{void} * \textbf{ArRecurrentTask::runThread} \; (\textbf{void} * \textit{ptr}) \\ & & [\texttt{virtual}] \end{array}$

The main run loop.

Override this function and put your taskes run loop here. Check the value of **getRunning**() (p. 488) or myRunning periodicly in your loop. If the value goes false, the loop should exit and **runThread**() (p. 324) should return.

Reimplemented from ArASyncTask (p. 111).

4.91.2.3 virtual void ArRecurrentTask::task () [pure virtual]

The main run loop.

Override this function and put your task here.

The documentation for this class was generated from the following files:

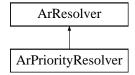
- ArRecurrentTask.h
- ArRecurrentTask.cpp

4.92 ArResolver Class Reference

Resolves a list of actions and returns what to do.

#include <ArResolver.h>

Inheritance diagram for ArResolver::



Public Types

• typedef std::multimap< int, **ArAction** *> **ActionMap****Constructor.

Public Methods

• virtual ~**ArResolver** ()

Desturctor.

• virtual ArActionDesired * resolve (ActionMap *actions, ArRobot *robot)=0

Figure out a single ArActionDesired (p. 51) from a list of ArAction (p. 39) s.

- virtual const char * **getName** (void) const
 - Gets the name of the resolver.
- virtual const char * **getDescription** (void) const Gets the long description fo the resolver.

4.92.1 Detailed Description

Resolves a list of actions and returns what to do.

This class exists just for resolve, which will always have to be overriden. The class is used to take a list of actions and find out what to do from that...

The documentation for this class was generated from the following file:

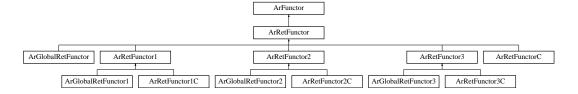
 \bullet ArResolver.h

4.93 ArRetFunctor Class Template Reference

Base class for functors with a return value.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor::



Public Methods

 $\bullet \ \, \mathrm{virtual} \sim \! \mathbf{ArRetFunctor} \,\, () \\$

Destructor.

- virtual void **invoke** (void)

 Invokes the functor.
- virtual Ret **invokeR** (void)=0

Invokes the functor with return value.

4.93.1 Detailed Description

$template < class \ Ret > \ class \ ArRetFunctor < \ Ret >$

Base class for functors with a return value.

This is the base class for functors with a return value. Code that has a reference to a functor that returns a value should use this class name. This allows the code to know how to invoke the functor without knowing which class the member function is in.

For an overall description of functors, see **ArFunctor** (p. 139).

The documentation for this class was generated from the following file:

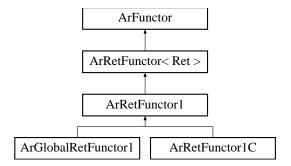
• ArFunctor.h

4.94 ArRetFunctor1 Class Template Reference

Base class for functors with a return value with 1 parameter.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor1::



Public Methods

- virtual ~**ArRetFunctor1** ()

 Destructor.
- virtual Ret invokeR (void)=0
 Invokes the functor with return value.
- virtual Ret invokeR (P1 p1)=0

 Invokes the functor with return value.

4.94.1 Detailed Description

template<class Ret, class P1> class ArRetFunctor1< Ret, P1>

Base class for functors with a return value with 1 parameter.

This is the base class for functors with a return value and take 1 parameter. Code that has a reference to a functor that returns a value and takes 1 parameter should use this class name. This allows the code to know how to invoke the functor without knowing which class the member function is in.

For an overall description of functors, see **ArFunctor** (p. 139).

4.94.2 Member Function Documentation

4.94.2.1 template<class Ret, class P1> virtual Ret ArRetFunctor1< Ret, P1>::invokeR (P1 p1) [pure virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented in **ArGlobalRetFunctor1** (p. 185), and **ArRetFunctor1C** (p. 332).

The documentation for this class was generated from the following file:

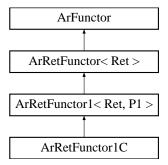
 \bullet ArFunctor.h

4.95 ArRetFunctor1C Class Template Reference

Functor for a member function with return value and 1 parameter.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor1C::



Public Methods

• ArRetFunctor1C ()

Constructor.

- ArRetFunctor1C (T &obj, Ret(T::*func)(P1))

 Constructor supply function pointer.
- ArRetFunctor1C (T &obj, Ret(T::*func)(P1), P1 p1)

 Constructor supply function pointer, default parameters.
- ArRetFunctor1C (T *obj, Ret(T::*func)(P1))

 Constructor supply function pointer.
- ArRetFunctor1C (T *obj, Ret(T::*func)(P1), P1 p1)

 Constructor supply function pointer, default parameters.
- virtual ~**ArRetFunctor1C** ()

 Destructor.
- virtual Ret **invokeR** (void)

 Invokes the functor with return value.

• virtual Ret invokeR (P1 p1)

Invokes the functor with return value.

- virtual void **setThis** (T *obj)

 Set the 'this' pointer.
- virtual void **setThis** (T &obj)

 Set the 'this' pointer.
- virtual void **setP1** (P1 p1)

 Set the default parameter.

4.95.1 Detailed Description

template<class Ret, class T, class P1> class ArRetFunctor1C< Ret, T, P1 >

Functor for a member function with return value and 1 parameter.

This is a class for member functions which take 1 parameter and return a value. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.95.2 Constructor & Destructor Documentation

4.95.2.1 template < class Ret, class T, class P1> ArRetFunctor1C < Ret, T, P1>::ArRetFunctor1C (T & obj, Ret(T::* func)(P1)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.95.2.2 template < class Ret, class T, class P1> ArRetFunctor1C < Ret, T, P1>::ArRetFunctor1C (T & obj, Ret(T::* func)(P1), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.95.2.3 template<class Ret, class T, class P1> ArRetFunctor1C< Ret, T, P1>::ArRetFunctor1C (T * obj, Ret(T::* func)(P1) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.95.2.4 template<class Ret, class T, class P1> ArRetFunctor1C< Ret, T, P1>::ArRetFunctor1C (T * obj, Ret(T::* func)(P1), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.95.3 Member Function Documentation

4.95.3.1 template<class Ret, class T, class P1> virtual Ret ArRetFunctor1C< Ret, T, P1>::invokeR (P1 p1) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented from ArRetFunctor1 (p. 329).

4.95.3.2 template<class Ret, class T, class P1> virtual void ArRetFunctor1C< Ret, T, P1>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.95.3.3 template<class Ret, class T, class P1> virtual void ArRetFunctor1C< Ret, T, P1>::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.95.3.4 template<class Ret, class T, class P1> virtual void ArRetFunctor1C< Ret, T, P1>::setThis (T * obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

The documentation for this class was generated from the following file:

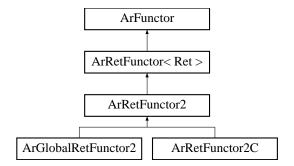
• ArFunctor.h

4.96 ArRetFunctor2 Class Template Reference

Base class for functors with a return value with 2 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor2::



Public Methods

- virtual ~**ArRetFunctor2** ()

 Destructor.
- virtual Ret invokeR (void)=0

 Invokes the functor with return value.
- virtual Ret invokeR (P1 p1)=0

 Invokes the functor with return value.
- virtual Ret **invokeR** (P1 p1, P2 p2)=0

 Invokes the functor with return value.

4.96.1 Detailed Description

template<class Ret, class P1, class P2> class ArRetFunctor2< Ret, P1, P2>

Base class for functors with a return value with 2 parameters.

This is the base class for functors with a return value and take 2 parameters. Code that has a reference to a functor that returns a value and takes 2 parameters should use this class name. This allows the code to know how to invoke the functor without knowing which class the member function is in.

For an overall description of functors, see **ArFunctor** (p. 139).

4.96.2 Member Function Documentation

4.96.2.1 template<class Ret, class P1, class P2> virtual Ret ArRetFunctor2< Ret, P1, P2>::invokeR (P1 p1, P2 p2) [pure virtual]

Invokes the functor with return value.

Parameters:

p1 first parameterp2 second parameter

Reimplemented in **ArGlobalRetFunctor2** (p. 189), and **ArRetFunctor2C** (p. 339).

4.96.2.2 template<class Ret, class P1, class P2> virtual Ret ArRetFunctor2< Ret, P1, P2>::invokeR (P1 p1) [pure virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented in **ArGlobalRetFunctor2** (p. 189), and **ArRetFunctor2C** (p. 339).

The documentation for this class was generated from the following file:

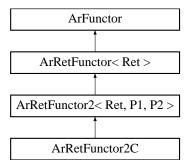
• ArFunctor.h

4.97 ArRetFunctor2C Class Template Reference

Functor for a member function with return value and 2 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor2C::



Public Methods

• ArRetFunctor2C ()

Constructor.

- ArRetFunctor2C (T &obj, Ret(T::*func)(P1, P2))

 Constructor supply function pointer.
- ArRetFunctor2C (T &obj, Ret(T::*func)(P1, P2), P1 p1)

 Constructor supply function pointer, default parameters.
- ArRetFunctor2C (T &obj, Ret(T::*func)(P1, P2), P1 p1, P2 p2)

 Constructor supply function pointer, default parameters.
- ArRetFunctor2C (T *obj, Ret(T::*func)(P1, P2))

 Constructor supply function pointer.
- ArRetFunctor2C (T *obj, Ret(T::*func)(P1, P2), P1 p1)

 Constructor supply function pointer, default parameters.
- ArRetFunctor2C (T *obj, Ret(T::*func)(P1, P2), P1 p1, P2 p2)

 Constructor supply function pointer, default parameters.

• virtual ~ArRetFunctor2C ()

Destructor.

• virtual Ret invokeR (void)

Invokes the functor with return value.

• virtual Ret invokeR (P1 p1)

Invokes the functor with return value.

• virtual Ret invokeR (P1 p1, P2 p2)

Invokes the functor with return value.

• virtual void **setThis** (T *obj)

 $Set\ the\ 'this'\ pointer.$

• virtual void **setThis** (T &obj)

Set the 'this' pointer.

• virtual void **setP1** (P1 p1)

Set the default parameter.

• virtual void setP2 (P2 p2)

Set the default 2nd parameter.

4.97.1 Detailed Description

template<class Ret, class T, class P1, class P2> class ArRet-Functor2C< Ret, T, P1, P2>

Functor for a member function with return value and 2 parameters.

This is a class for member functions which take 2 parameters and return a value. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.97.2 Constructor & Destructor Documentation

4.97.2.1 template<class Ret, class T, class P1, class P2> ArRetFunctor2C< Ret, T, P1, P2>::ArRetFunctor2C (T & obj, Ret(T::* func)(P1, P2)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.97.2.2 template<class Ret, class T, class P1, class P2>
ArRetFunctor2C< Ret, T, P1, P2>::ArRetFunctor2C (T
& obj, Ret(T::* func)(P1, P2), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer p1 default first parameter

4.97.2.3 template<class Ret, class T, class P1, class P2> ArRetFunctor2C< Ret, T, P1, P2>::ArRetFunctor2C (T & obj, Ret(T::* func)(P1, P2), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointerp1 default first parameterp2 default second parameter

4.97.2.4 template<class Ret, class T, class P1, class P2> ArRetFunctor2C< Ret, T, P1, P2>::ArRetFunctor2C (T * obj, Ret(T::* func)(P1, P2)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.97.2.5 template<class Ret, class T, class P1, class P2>
ArRetFunctor2C< Ret, T, P1, P2>::ArRetFunctor2C (T
* obj, Ret(T::* func)(P1, P2), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.97.2.6 template<class Ret, class T, class P1, class P2> ArRetFunctor2C< Ret, T, P1, P2>::ArRetFunctor2C (T * obj, Ret(T::* func)(P1, P2), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointerp1 default first parameterp2 default second parameter

4.97.3 Member Function Documentation

4.97.3.1 template<class Ret, class T, class P1, class P2> virtual Ret ArRetFunctor2C< Ret, T, P1, P2>::invokeR (P1 p1, P2 p2) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameterp2 second parameter

Reimplemented from **ArRetFunctor2** (p. 335).

4.97.3.2 template<class Ret, class T, class P1, class P2> virtual Ret ArRetFunctor2C< Ret, T, P1, P2>::invokeR (P1 p1) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented from ArRetFunctor2 (p. 335).

4.97.3.3 template<class Ret, class T, class P1, class P2> virtual void ArRetFunctor2C< Ret, T, P1, P2>::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.97.3.4 template<class Ret, class T, class P1, class P2> virtual void ArRetFunctor2C< Ret, T, P1, P2>::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

4.97.3.5 template<class Ret, class T, class P1, class P2> virtual void ArRetFunctor2C< Ret, T, P1, P2>::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.97.3.6 template<class Ret, class T, class P1, class P2> virtual void ArRetFunctor2C< Ret, T, P1, P2>::setThis (T * obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

The documentation for this class was generated from the following file:

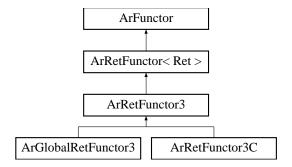
• ArFunctor.h

4.98 ArRetFunctor3 Class Template Reference

Base class for functors with a return value with 3 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor3::



Public Methods

- virtual ~**ArRetFunctor3** ()

 Destructor.
- virtual Ret invokeR (void)=0

 Invokes the functor with return value.
- virtual Ret invokeR (P1 p1)=0

 Invokes the functor with return value.
- virtual Ret **invokeR** (P1 p1, P2 p2)=0

 Invokes the functor with return value.
- virtual Ret **invokeR** (P1 p1, P2 p2, P3 p3)=0

 Invokes the functor with return value.

4.98.1 Detailed Description

template<class Ret, class P1, class P2, class P3> class ArRet-Functor3< Ret, P1, P2, P3>

Base class for functors with a return value with 3 parameters.

This is the base class for functors with a return value and take 3 parameters. Code that has a reference to a functor that returns a value and takes 3 parameters should use this class name. This allows the code to know how to invoke the functor without knowing which class the member function is in.

For an overall description of functors, see **ArFunctor** (p. 139).

4.98.2 Member Function Documentation

4.98.2.1 template<class Ret, class P1, class P2, class P3> virtual Ret ArRetFunctor3< Ret, P1, P2, P3>::invokeR (P1 p1, P2 p2, P3 p3) [pure virtual]

Invokes the functor with return value.

Parameters:

```
p1 first parameter
```

p2 second parameter

p3 third parameter

Reimplemented in **ArGlobalRetFunctor3** (p. 193), and **ArRetFunctor3C** (p. 349).

4.98.2.2 template<class Ret, class P1, class P2, class P3> virtual Ret ArRetFunctor3< Ret, P1, P2, P3>::invokeR (P1 p1, P2 p2) [pure virtual]

Invokes the functor with return value.

Parameters:

```
p1 first parameter
```

p2 second parameter

Reimplemented in **ArGlobalRetFunctor3** (p. 194), and **ArRetFunctor3C** (p. 349).

4.98.2.3 template<class Ret, class P1, class P2, class P3> virtual Ret ArRetFunctor3< Ret, P1, P2, P3>::invokeR (P1 p1) [pure virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented in **ArGlobalRetFunctor3** (p. 194), and **ArRetFunctor3C** (p. 350).

The documentation for this class was generated from the following file:

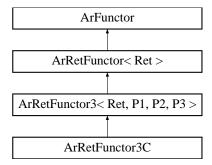
 \bullet Ar
Functor.h

4.99 ArRetFunctor3C Class Template Reference

Functor for a member function with return value and 3 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor3C::



Public Methods

• ArRetFunctor3C ()

Constructor.

- ArRetFunctor3C (T &obj, Ret(T::*func)(P1, P2, P3))

 Constructor supply function pointer.
- ArRetFunctor3C (T &obj, Ret(T::*func)(P1, P2, P3), P1 p1)

 Constructor supply function pointer, default parameters.
- ArRetFunctor3C (T &obj, Ret(T::*func)(P1, P2, P3), P1 p1, P2 p2)

 Constructor supply function pointer, default parameters.
- ArRetFunctor3C (T &obj, Ret(T::*func)(P1, P2, P3), P1 p1, P2 p2, P3 p3)

Constructor - supply function pointer, default parameters.

- ArRetFunctor3C (T *obj, Ret(T::*func)(P1, P2, P3))

 Constructor supply function pointer.
- ArRetFunctor3C (T *obj, Ret(T::*func)(P1, P2, P3), P1 p1)

 Constructor supply function pointer, default parameters.

- ArRetFunctor3C (T *obj, Ret(T::*func)(P1, P2, P3), P1 p1, P2 p2)

 Constructor supply function pointer, default parameters.
- **ArRetFunctor3C** (T *obj, Ret(T::*func)(P1, P2, P3), P1 p1, P2 p2, P3 p3)

 $Constructor \hbox{ - } supply \hbox{ function pointer, default parameters.}$

• virtual ~ArRetFunctor3C ()

Destructor.

- virtual Ret invokeR (void)
 Invokes the functor with return value.
- virtual Ret invokeR (P1 p1)
 Invokes the functor with return value.
- virtual Ret invokeR (P1 p1, P2 p2)

 Invokes the functor with return value.
- virtual Ret **invokeR** (P1 p1, P2 p2, P3 p3)

 Invokes the functor with return value.
- virtual void **setThis** (T *obj)

 Set the 'this' pointer.
- virtual void **setThis** (T &obj)

 Set the 'this' pointer.
- virtual void **setP1** (P1 p1)

 Set the default parameter.
- virtual void **setP2** (P2 p2)

 Set the default 2nd parameter.
- virtual void **setP3** (P3 p3)

 Set the default third parameter.

4.99.1 Detailed Description

template<class Ret, class T, class P1, class P2, class P3> class Ar-RetFunctor3C< Ret, T, P1, P2, P3>

Functor for a member function with return value and 3 parameters.

This is a class for member functions which take 3 parameters and return a value. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.99.2 Constructor & Destructor Documentation

4.99.2.1 template<class Ret, class T, class P1, class P2, class P3>
ArRetFunctor3C< Ret, T, P1, P2, P3>::ArRetFunctor3C
(T & obj, Ret(T::* func)(P1, P2, P3)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.99.2.2 template<class Ret, class T, class P1, class P2, class P3>
ArRetFunctor3C< Ret, T, P1, P2, P3>::ArRetFunctor3C
(T & obj, Ret(T::* func)(P1, P2, P3), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer
p1 default first parameter

4.99.2.3 template<class Ret, class T, class P1, class P2, class P3> ArRetFunctor3C< Ret, T, P1, P2, P3>::ArRetFunctor3C (T & obj, Ret(T::* func)(P1, P2, P3), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer

```
p1 default first parameter p2 default second parameter
```

4.99.2.4 template < class Ret, class T, class P1, class P2, class P3>
ArRetFunctor3C < Ret, T, P1, P2, P3 >::ArRetFunctor3C
(T & obj, Ret(T::* func)(P1, P2, P3), P1 p1, P2 p2, P3
p3) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointerp1 default first parameterp2 default second parameter

4.99.2.5 template<class Ret, class T, class P1, class P2, class P3> ArRetFunctor3C< Ret, T, P1, P2, P3>::ArRetFunctor3C (T*obj, Ret(T::*func)(P1, P2, P3)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.99.2.6 template<class Ret, class T, class P1, class P2, class P3> ArRetFunctor3C< Ret, T, P1, P2, P3>::ArRetFunctor3C (T*obj, Ret(T::*func)(P1, P2, P3), P1 p1) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

func member function pointer p1 default first parameter

4.99.2.7 template<class Ret, class T, class P1, class P2, class P3> ArRetFunctor3C< Ret, T, P1, P2, P3>::ArRetFunctor3C (T*obj, Ret(T::*func)(P1, P2, P3), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

```
func member function pointerp1 default first parameterp2 default second parameter
```

Constructor - supply function pointer, default parameters.

Parameters:

```
func member function pointer
p1 default first parameter
p2 default second parameter
p3 default third parameter
```

4.99.3 Member Function Documentation

4.99.3.1 template<class Ret, class T, class P1, class P2, class P3> virtual Ret ArRetFunctor3C< Ret, T, P1, P2, P3 >::invokeR (P1 p1, P2 p2, P3 p3) [inline, virtual]

Invokes the functor with return value.

Parameters:

```
p1 first parameter p2 second parameter
```

Reimplemented from **ArRetFunctor3** (p. 343).

4.99.3.2 template<class Ret, class T, class P1, class P2, class P3> virtual Ret ArRetFunctor3C< Ret, T, P1, P2, P3 >::invokeR (P1 p1, P2 p2) [inline, virtual]

Invokes the functor with return value.

Parameters:

```
p1 first parameterp2 second parameter
```

Reimplemented from **ArRetFunctor3** (p. 343).

4.99.3.3 template<class Ret, class T, class P1, class P2, class P3> virtual Ret ArRetFunctor3C< Ret, T, P1, P2, P3 >::invokeR (P1 p1) [inline, virtual]

Invokes the functor with return value.

Parameters:

p1 first parameter

Reimplemented from **ArRetFunctor3** (p. 343).

4.99.3.4 template<class Ret, class T, class P1, class P2, class P3> virtual void ArRetFunctor3C< Ret, T, P1, P2, P3 >::setP1 (P1 p1) [inline, virtual]

Set the default parameter.

Parameters:

p1 default first parameter

4.99.3.5 template<class Ret, class T, class P1, class P2, class P3> virtual void ArRetFunctor3C< Ret, T, P1, P2, P3 >::setP2 (P2 p2) [inline, virtual]

Set the default 2nd parameter.

Parameters:

p2 default second parameter

4.99.3.6 template < class Ret, class T, class P1, class P2, class P3> virtual void ArRetFunctor3C < Ret, T, P1, P2, P3 >::setP3 (P3 p3) [inline, virtual]

Set the default third parameter.

Parameters:

p3 default third parameter

4.99.3.7 template<class Ret, class T, class P1, class P2, class P3> virtual void ArRetFunctor3C< Ret, T, P1, P2, P3 >::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.99.3.8 template < class Ret, class T, class P1, class P2, class P3> virtual void ArRetFunctor3C < Ret, T, P1, P2, P3 >::setThis (T * obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

The documentation for this class was generated from the following file:

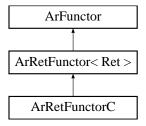
• ArFunctor.h

4.100 ArRetFunctorC Class Template Reference

Functor for a member function with return value.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctorC::



Public Methods

• ArRetFunctorC ()

Constructor.

- ArRetFunctorC (T &obj, Ret(T::*func)(void))

 Constructor supply function pointer.
- ArRetFunctorC (T *obj, Ret(T::*func)(void))

 Constructor supply function pointer.
- virtual ~ArRetFunctorC ()

 Destructor supply function pointer.
- virtual Ret invokeR (void)
 Invokes the functor with return value.
- virtual void **setThis** (T *obj)

 Set the 'this' pointer.
- virtual void **setThis** (T &obj)

 Set the 'this' pointer.

4.100.1 Detailed Description

template < class Ret, class T > class ArRetFunctorC < Ret, T >

Functor for a member function with return value.

This is a class for member functions which return a value. This class contains the knowledge on how to call a member function on a particular instance of a class. This class should be instantiated by code that wishes to pass off a functor to another piece of code.

For an overall description of functors, see **ArFunctor** (p. 139).

4.100.2 Constructor & Destructor Documentation

4.100.2.1 template<class Ret, class T> ArRetFunctorC< Ret, T >::ArRetFunctorC (T & obj, Ret(T::* func)(void))
[inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.100.2.2 template<class Ret, class T> ArRetFunctorC< Ret, T>::ArRetFunctorC (T * obj, Ret(T::* func)(void)) [inline]

Constructor - supply function pointer.

Parameters:

func member function pointer

4.100.3 Member Function Documentation

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

4.100.3.2 template<class Ret, class T> virtual void ArRetFunctorC< Ret, T>::setThis (T * obj) [inline, virtual]

Set the 'this' pointer.

Parameters:

obj the 'this' pointer

The documentation for this class was generated from the following file:

 \bullet Ar
Functor.h

4.101 ArRobot Class Reference

THE important class.

#include <ArRobot.h>

Public Types

• enum WaitState { WAIT_CONNECTED, WAIT_FAILED_-CONN, WAIT_RUN_EXIT, WAIT_TIMEDOUT, WAIT_INTR, WAIT_FAIL }

Public Methods

• **ArRobot** (const char *name=NULL, bool doStateReflection=true, bool doSigHandle=true, bool normalInit=true)

Constructor.

• \sim ArRobot ()

Destructor.

• void **run** (bool stopRunIfNotConnected)

Starts the instance to do processing in this thread.

• void runAsync (bool stopRunIfNotConnected)

Starts the instance to do processing in its own new thread.

• bool isRunning (void) const

Returns whether the robot is currently running or not.

• void **stopRunning** (bool doDisconnect=true)

Stops the robot from doing any more processing.

• void **setDeviceConnection** (**ArDeviceConnection** *connection)

Sets the connection this instance uses.

• ArDeviceConnection * getDeviceConnection (void) const

Gets the connection this instance uses.

• bool isConnected (void) const

Questions whether the robot is connected or not.

• bool blockingConnect (void)

Connects to a robot, not returning until connection made or failed.

• bool asyncConnect (void)

Connects to a robot, from the robots own thread.

• bool disconnect (void)

Disconnects from a robot.

• void **clearDirectMotion** (void)

Clears what direct motion commands have been given, so actions work.

• bool isDirectMotion (void) const

Returns true if direct motion commands are blocking actions.

• void enableMotors ()

Enables the motors on the robot.

• void **disableMotors** ()

Disables the motors on the robot.

• void **stop** (void)

Stops the robot See also: clearDirectMotion (p. 380).

• void **setVel** (double velocity)

Sets the velocity
See also:
clearDirectMotion (p. 380).

• void **setVel2** (double leftVelocity, double rightVelocity)

Sets the velocity of the wheels independently See also: clearDirectMotion (p. 380).

• void **move** (double distance)

Move the given distance forward/backwards See also: clearDirectMotion (p. 380).

• bool isMoveDone (double delta=0.0)

Sees if the robot is done moving the previously given move.

• void **setMoveDoneDist** (double dist)

Sets the difference required for being done with a move.

• double **getMoveDoneDist** (void)

Gets the difference required for being done with a move.

• void **setHeading** (double heading)

```
Sets the heading
See also:
clearDirectMotion (p. 380).
```

• void **setRotVel** (double velocity)

```
Sets the rotational velocity
See also:
clearDirectMotion (p. 380).
```

• void **setDeltaHeading** (double deltaHeading)

```
Sets the delta heading
See also:
clearDirectMotion (p. 380).
```

• bool isHeadingDone (double delta=0.0) const

Sees if the robot is done changing to the previously given setHeading.

• void **setHeadingDoneDiff** (double degrees)

sets the difference required for being done with a heading change.

• double **getHeadingDoneDiff** (void) const

Gets the difference required for being done with a heading change.

• void **setDirectMotionPrecedenceTime** (int mSec)

Sets the length of time a direct motion command will take precedence over actions, in milliseconds.

• unsigned int **getDirectMotionPrecedenceTime** (void) const

Gets the length of time a direct motion command will take precedence over actions, in milliseconds.

• bool **com** (unsigned char command)

Sends a command to the robot with no arguments.

• bool **comInt** (unsigned char command, short int argument)

Sends a command to the robot with an int for argument.

- bool **com2Bytes** (unsigned char command, char high, char low)

 Sends a command to the robot with two bytes for argument.
- bool **comStr** (unsigned char command, const char *argument)

 Sends a command to the robot with a string for argument.
- bool **comStrN** (unsigned char command, const char *str, int size)

 Sends a command to the robot with a size bytes of str as argument.
- const char * **getRobotName** (void) const

 Returns the Robot's name that is set in its onboard configuration.
- const char * **getRobotType** (void) const Returns the type of the robot connected to.
- const char * getRobotSubType (void) const Returns the subtype of the robot connected to.
- double getMaxTransVel (void) const
 Gets the robots maximum translational velocity.
- bool **setMaxTransVel** (double maxVel)

 Sets the robots maximum translational velocity.
- double **getMaxRotVel** (void) const

 Gets the robots maximum rotational velocity.
- bool **setMaxRotVel** (double myMaxVel)

 Sets the robots maximum rotational velocity.
- ArPose getPose (void) const Gets the global position of the robot.
- double getX (void) const
 Gets the global X location of the robot.
- double **getY** (void) const

 Gets the global Y location of the robot.
- double **getTh** (void) const

Gets the global Th location of the robot.

- double **getVel** (void) const

 Gets the translational velocity of the robot.
- double **getRotVel** (void) const

 Gets the rotational velocity of the robot.
- double **getRobotRadius** (void) const Gets the robot radius (in mm).
- double **getRobotDiagonal** (void) const

 Gets the robot diagonal (half-height to diagonal of octagon) (in mm).
- double **getBatteryVoltage** (void) const Gets the battery voltage of the robot.
- double **getLeftVel** (void) const Gets the velocity of the left wheel.
- double **getRightVel** (void) const Gets the velocity of the right wheel.
- int getStallValue (void) const

 Gets the 2 bytes of stall return from the robot.
- bool isLeftMotorStalled (void) const Returns true if the left motor is stalled.
- bool **isRightMotorStalled** (void) const Returns true if the left motor is stalled.
- double **getControl** (void) const Gets the control heading.
- int **getFlags** (void) const Gets the flags values.
- bool areMotorsEnabled (void) const returns true if the motors are enabled.
- bool areSonarsEnabled (void) const

returns true if the motors are enabled.

- double **getCompass** (void) const Gets the compass heading from the robot.
- int getAnalogPortSelected (void) const Gets which analog port is selected.
- unsigned char **getAnalog** (void) const Gets the analog value.
- unsigned char **getDigIn** (void) const Gets the byte representing digital input status.
- unsigned char **getDigOut** (void) const Gets the byte representing digital output status.
- int getIOAnalogSize (void) const Gets the number of bytes in the analog IO buffer.
- int **getIODigInSize** (void) const

 Gets the number of bytes in the digital input IO buffer.
- int getIODigOutSize (void) const

 Gets the number of bytes in the digital output IO buffer.
- int getIOAnalog (int num) const

 Gets the n'th byte from the analog input data from the IO packet.
- unsigned char **getIODigIn** (int num) const Gets the n'th byte from the digital input data from the IO packet.
- unsigned char **getIODigOut** (int num) const

 Gets the n'th byte from the digital output data from the IO packet.
- bool hasTableSensingIR (void) const

 Gets whether the robot has table sensing IR or not (see params in docs).
- bool isLeftTableSensingIRTriggered (void) const Returns true if the left table sensing IR is triggered.
- bool isRightTableSensingIRTriggered (void) const

Returns true if the right table sensing IR is triggered.

ullet bool isLeftBreakBeamTriggered (void) const

Returns true if the left break beam IR is triggered.

• bool isRightBreakBeamTriggered (void) const

Returns true if the right break beam IR is triggered.

• ArTime getIOPacketTime (void) const

Returns the time received of the last IO packet.

• bool hasFrontBumpers (void) const

Gets whether the robot has front bumpers (see params in docs).

• unsigned int **getNumFrontBumpers** (void) const

Gets the number of the front bumpers.

• bool hasRearBumpers (void) const

Gets whether the robot has rear bumpers (see params in docs).

• unsigned int **getNumRearBumpers** (void) const

Gets the number of the rear bumpers.

• ArPose getEncoderPose (void) const

Gets the position of the robot according to the encoders.

• int getMotorPacCount (void) const

Gets the number of motor packets received in the last second.

• int getSonarPacCount (void) const

Gets the number of sonar returns received in the last second.

• int getSonarRange (int num) const

Gets the range of the last sonar reading for the given sonar.

• bool **isSonarNew** (int num) const

Find out if the given sonar has a new reading.

• int getNumSonar (void) const

Find the number of sonar there are.

• ArSensorReading * getSonarReading (int num) const

Returns the sonar reading for the given sonar.

- int **getClosestSonarRange** (double startAngle, double endAngle) const Returns the closest of the current sonar reading in the given range.
- int **getClosestSonarNumber** (double startAngle, double endAngle) const

Returns the number of the sonar that has the closest current reading in the given range.

- const char * **getName** (void) const Gets the robots name in ARIAs list.
- void **setName** (const char *name)

 Sets the robots name in ARIAs list.
- void **moveTo** (**ArPose** pose, bool doCumulative=true)

 Moves the robot's idea of its position to this position.
- void **moveTo** (**ArPose** to, **ArPose** from, bool doCumulative=true)

 Moves the robot's RW position to reflect pose From => pose To.
- void **setEncoderTransform** (**ArPose** deadReconPos, **ArPose** global-Pos)

Changes the transform.

- void **setEncoderTransform** (**ArPose** transformPos)

 Changes the transform directly.
- ArTransform getEncoderTransform (void) const Gets the encoder transform.
- ArTransform getToGlobalTransform (void) const This gets the transform from local coords to global coords.
- ArTransform getToLocalTransform (void) const

 This gets the transform for going from global coords to local coords.
- void **applyTransform** (**ArTransform** trans, bool do-Cumulative=true)

This applies a transform to all the robot range devices and to the sonar.

• void setDeadReconPose (ArPose pose)

Sets the dead recon position of the robot.

• void addRangeDevice (ArRangeDevice *device)

Adds a rangeDevice to the robot's list of them, and set the device's robot pointer.

• void remRangeDevice (const char *name)

Remove a range device from the robot's list, by name.

• void remRangeDevice (ArRangeDevice *device)

Remove a range device from the robot's list, by instance.

• const ArRangeDevice * findRangeDevice (const char *name) const

Finds a rangeDevice in the robot's list.

• ArRangeDevice * findRangeDevice (const char *name)

Finds a rangeDevice in the robot's list.

• std::list< **ArRangeDevice** *> * **getRangeDeviceList** (void)

Gets the range device list.

• bool hasRangeDevice (ArRangeDevice *device) const

Finds whether a particular range device is attached to this robot or not.

• double **checkRangeDevicesCurrentPolar** (double startAngle, double endAngle, double *angle=NULL) const

Goes through all the range devices and checks them.

• double **checkRangeDevicesCumulativePolar** (double startAngle, double endAngle, double *angle=NULL) const

Goes through all the range devices and checks them.

- double **checkRangeDevicesCurrentBox** (double x1, double y1, double x2, double y2, **ArPose** *readingPos=NULL) const
- double **checkRangeDevicesCumulativeBox** (double x1, double y1, double x2, double y2, **ArPose** *readingPos=NULL) const
- void **setStateReflectionRefreshTime** (int msec)

Sets the number of milliseconds between state reflection refreshes if the state has not changed.

• int getStateReflectionRefreshTime (void) const

Sets the number of milliseconds between state reflection refreshes if the state has not changed.

void addPacketHandler (ArRetFunctor1< bool, ArRobotPacket
 *> *functor, ArListPos::Pos position)

Adds a packet handler to the list of packet handlers.

void remPacketHandler (ArRetFunctor1< bool, ArRobotPacket
 *> *functor)

Removes a packet handler from the list of packet handlers.

• void addConnectCB (ArFunctor *functor, ArListPos::Pos position)

Adds a connect callback.

• void remConnectCB (ArFunctor *functor)

Adds a disconnect callback.

• void addFailedConnectCB (ArFunctor *functor, ArListPos::Pos position)

Adds a callback for when a connection to the robot is failed.

• void remFailedConnectCB (ArFunctor *functor)

Removes a callback for when a connection to the robot is failed.

• void addDisconnectNormallyCB (ArFunctor *functor, ArList-Pos::Pos position)

Adds a callback for when disconnect is called while connected.

• void remDisconnectNormallyCB (ArFunctor *functor)

 $Removes\ a\ callback\ for\ when\ disconnect\ is\ called\ while\ connected.$

• void addDisconnectOnErrorCB (ArFunctor *functor, ArList-Pos::Pos position)

Adds a callback for when disconnection happens because of an error.

• void remDisconnectOnErrorCB (ArFunctor *functor)

 $Removes\ a\ callback\ for\ when\ disconnection\ happens\ because\ of\ an\ error.$

void addRunExitCB (ArFunctor *functor, ArListPos::Pos position)

Adds a callback for when the run loop exits for what ever reason.

• void remRunExitCB (ArFunctor *functor)

Removes a callback for when the run loop exits for what ever reason.

• WaitState waitForConnect (unsigned int msecs=0)

Suspend calling thread until the ArRobot is connected.

• WaitState waitForConnectOrConnFail (unsigned int msecs=0)

Suspend calling thread until the ArRobot is connected or fails to connect.

• WaitState waitForRunExit (unsigned int msecs=0)

Suspend calling thread until the ArRobot run loop has exited.

• void wakeAllWaitingThreads ()

Wake up all threads waiting on this robot.

• void wakeAllConnWaitingThreads ()

Wake up all threads waiting for connection.

• void wakeAllConnOrFailWaitingThreads ()

Wake up all threads waiting for connection or connection failure.

• void wakeAllRunExitWaitingThreads ()

Wake up all threads waiting for the run loop to exit.

• bool addUserTask (const char *name, int position, ArFunctor *functor, ArTaskState::State *state=NULL)

Adds a user task to the list of synchronous taskes.

• void remUserTask (const char *name)

Removes a user task from the list of synchronous taskes by name.

• void remUserTask (ArFunctor *functor)

Removes a user task from the list of synchronous taskes by functor.

• ArSyncTask * findUserTask (const char *name)

Finds a user task by name.

• ArSyncTask * findUserTask (ArFunctor *functor)

Finds a user task by functor.

• void logUserTasks (void) const

Logs the list of user tasks, strictly for your viewing pleasure.

• void logAllTasks (void) const

Logs the list of all tasks, strictly for your viewing pleasure.

• bool addSensorInterpTask (const char *name, int position, ArFunctor *functor, ArTaskState::State *state=NULL)

Adds a task under the sensor interp part of the syncronous tasks.

• void remSensorInterpTask (const char *name)

Removes a sensor interp tasks by name.

• void remSensorInterpTask (ArFunctor *functor)

Removes a sensor interp tasks by functor.

• ArSyncTask * findTask (const char *name)

Finds a task by name.

• ArSyncTask * findTask (ArFunctor *functor)

Finds a task by functor.

• void addAction (ArAction *action, int priority)

Adds an action to the list with the given priority.

• bool remAction (ArAction *action)

Removes an action from the list, by pointer.

• bool **remAction** (const char *actionName)

Removes an action from the list, by name.

• ArAction * findAction (const char *actionName)

Returns the first (highest priority) action with the given name (or NULL).

• ArResolver::ActionMap * getActionMap (void)

Returns the map of actions... don't do this unless you really know what you're doing.

• void logActions (void) const

Logs out the actions and their priorities.

• ArResolver * getResolver (void)

Gets the resolver the robot is using.

• void **setResolver** (**ArResolver** *resolver)

Sets the resolver the robot is using.

void setEncoderCorrectionCallback (ArRetFunctor1< double, Ar-PoseWithTime > *functor)

Sets the encoderCorrectionCallback.

 ArRetFunctor1< double, ArPoseWithTime > * getEncoder-CorrectionCallback (void) const

 $Gets\ the\ encoder Correction Callback.$

• void **setCycleTime** (unsigned int ms)

Sets the number of ms between cycles.

• unsigned int **getCycleTime** (void) const

Gets the number of ms between cycles.

• void **setConnectionCycleMultiplier** (unsigned int multiplier)

Sets the multiplier for how many cycles ArRobot waits when connecting.

• unsigned int **getConnectionCycleMultiplier** (void) const

Gets the multiplier for how many cycles ArRobot waits when connecting.

• void **setCycleChained** (bool cycleChained)

Sets whether to chain the robot cycle to when we get in SIP packets.

• bool **isCycleChained** (void) const

Gets whether we chain the robot cycle to when we get in SIP packets.

• void **setConnectionTimeoutTime** (int mSecs)

Sets the time without a response until connection assumed lost.

• int getConnectionTimeoutTime (void) const

Gets the time without a response until connection assumed lost.

• ArTime getLastPacketTime (void) const

Gets the time the last packet was received.

• void **setPoseInterpNumReadings** (size_t numReadings)

Sets the number of packets back in time the ArInterpolation (p. 210) goes.

• size_t getPoseInterpNumReadings (void) const

Sets the number of packets back in time the position interpol goes.

• int getPoseInterpPosition (ArTime timeStamp, ArPose *position)

Gets the position the robot was at at the given timestamp.

• unsigned int **getCounter** (void) const

Gets the Counter for the time through the loop.

• const ArRobotParams * getRobotParams (void) const

Gets the parameters the robot is using.

• bool loadParamFile (const char *file)

Loads a parameter file (replacing all other params).

• void **attachKeyHandler** (**ArKeyHandler** *keyHandler, bool exitOn-Escape=true)

Attachs a key handler.

• ArKeyHandler * getKeyHandler (void) const

Gets the key handler attached to this robot.

• int lock ()

Lock the robot instance.

• int **tryLock** ()

Try to lock the robot instance without blocking.

• int unlock ()

Unlock the robot instance.

ArSyncTask * getSyncTaskRoot (void)

This gets the root of the syncronous task tree, only serious developers should use it.

• void **loopOnce** (void)

This function loops once... only serious developers should use it.

• void incCounter (void)

This is only for use by syncLoop.

• void packetHandler (void)

Packet Handler, internal.

• void actionHandler (void)

Action Handler, internal.

• void stateReflector (void)

State Reflector, internal.

• void robotLocker (void)

Robot locker, internal.

• void robotUnlocker (void)

Robot unlocker, internal.

• void keyHandlerExit (void)

For the key handler, escape calls this to exit, internal.

• bool processMotorPacket (ArRobotPacket *packet)

Processes a motor packet, internal.

• void **processNewSonar** (char number, int range, **ArTime** time-Received)

Processes a new sonar reading, internal.

• bool processEncoderPacket (ArRobotPacket *packet)

Processes a new encoder packet, internal.

• bool **processIOPacket** (**ArRobotPacket** *packet)

Processes a new IO packet, internal.

• void **init** (void)

Internal function, shouldn't be used.

• void setUpSyncList (void)

Internal function, shouldn't be used, sets up the default sync list.

• void setUpPacketHandlers (void)

Internal function, shouldn't be used, sets up the default packet handlers.

• int asyncConnectHandler (bool tryHarderToConnect)

Internal function, shouldn't be used, does a single run of connecting.

• void **dropConnection** (void)

Internal function, shouldn't be used, drops the conn because of error.

• void **failedConnect** (void)

Internal function, shouldn't be used, denotes the conn failed.

• void madeConnection (void)

Internal function, shouldn't be used, does the after conn stuff.

• bool handlePacket (ArRobotPacket *packet)

Internal function, takes a packet and passes it to the packet handlers, returns true if handled, false otherwise.

• std::list < ArFunctor *> * getRunExitListCopy ()

Internal function, shouldn't be used, does what its name says.

• void **processParamFile** (ArRobotParamFile *paramFile)

Internal function, processes a parameter file.

4.101.1 Detailed Description

THE important class.

This is the most important class, the only classes most people will ever have to use are this one, and the **ArSerialConnection** (p. 424) and ArTCPConnection. NOTE: In Windows you cannot make an ArRobot a global, it will crash because the windows compiler initializes the constructors in the wrong order... you can make a pointer to an ArRobot and then new one however.

See also:

ArSerialConnection (p. 424), ArTcpConnection (p. 481)

4.101.2 Member Enumeration Documentation

4.101.2.1 enum ArRobot::WaitState

Enumeration values:

WAIT_CONNECTED The robot has connected.

WAIT_FAILED_CONN The robot failed to connect.

WAIT_RUN_EXIT The run loop has exited.

WAIT_TIMEDOUT The wait reached the timeout specified.

WAIT_INTR The wait was interupted by a signal.

WAIT_FAIL The wait failed due to an error.

4.101.3 Constructor & Destructor Documentation

4.101.3.1 ArRobot::ArRobot (const char * name = NULL, bool doStateReflection = true, bool doSigHandle = true, bool normalInit = true)

Constructor.

Parameters:

doStateReflection whether the robot should use direct motion command reflection or simply send commands when it gets them. If state-Reflection is on then when one of the movement commands is given it stores the value and then sends it at the appropriate spot in the cycle. If stateReflection isn't on, then a pulse will be sent every other cycle to make sure the watchdog timer doesn't kick off.

normalInit whether the robot should initialize its structures or the calling program will take care of it. No one will probably ever use this value, since if they are doing that then overriding will probably be more useful, but there it is.

doSigHandle do normal signal handling and have this robot instance stopRunning() (p. 403) when the program is signaled

4.101.4 Member Function Documentation

4.101.4.1 void ArRobot::actionHandler (void)

Action Handler, internal.

Runs the resolver on the actions, if state reflection (direct motion reflection really) is enabled in the **ArRobot::ArRobot** (p. 371) constructor then it just saves these values for use by the stateReflector, otherwise it sends these values straight down to the robot.

See also:

addAction (p. 371), remAction (p. 393)

4.101.4.2 void ArRobot::addAction (ArAction * action, int priority)

Adds an action to the list with the given priority.

Adds an action to the list of actions at the given priority, in the case of two (or more) actions with the same priority, the default resolver (ArPriorityResolver (p. 297)) averages the the multiple readings... the priority can be any integer, but as a convention 0 to 100 is used, with 100 being the highest priority.

Parameters:

action the action to add
priority what importance to give the action

4.101.4.3 void ArRobot::addConnectCB (ArFunctor * functor, ArListPos::Pos position)

Adds a connect callback.

Adds a connect callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). The entire list of connect callbacks is called when a connection is made with the robot. If you have some sort of module that adds a callback, that module must remove the callback when the module is removed.

Parameters:

functorfunctor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remConnectCB (p. 393)

4.101.4.4 void ArRobot::addDisconnectNormallyCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when disconnect is called while connected.

Adds a disconnect normally callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). This whole list of disconnect normally callbacks is called when something calls disconnect if the instance is Connected. If there is no connection and disconnect is called nothing is done. If you have some sort of module that adds a callback, that module must remove the callback when the module is removed.

Parameters:

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remFailedConnectCB (p. 394)

4.101.4.5 void ArRobot::addDisconnectOnErrorCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when disconnection happens because of an error.

Adds a disconnect on error callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). This whole list of disconnect on error callbacks is called when ARIA loses connection to a robot because of an error. This can occur if the physical connection (ie serial cable) between the robot and the computer is severed/disconnected, if one of a pair of radio modems that connect the robot and computer are disconnected, if someone presses the reset button on the robot, or if the simulator is closed while ARIA is connected to it. Note that if the link between the two is lost the ARIA assumes it is temporary until it reaches a timeout value set with setConnectionTimeoutTime. If you have some sort of module that adds a callback, that module must remove the callback when the module removed.

Parameters:

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remFailedConnectCB (p. 394)

4.101.4.6 void ArRobot::addFailedConnectCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when a connection to the robot is failed.

Adds a failed connect callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). This whole list of failed connect callbacks is called when an attempt is made to connect to the robot, but fails. The usual reason for this failure is either that there is no robot/sim where the connection was tried to be made, the robot wasn't given a connection, or the radio modems that communicate with the robot aren't on. If you have some sort of module that adds a callback, that module must remove the callback when the module removed.

Parameters:

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remFailedConnectCB (p. 394)

4.101.4.7 void ArRobot::addPacketHandler (ArRetFunctor1< bool, ArRobotPacket *> * functor, ArListPos::Pos position)

Adds a packet handler to the list of packet handlers.

Adds a packet handler. A packet handler is an **ArRetFunctor1** (p. 328), created as an instance of **ArRetFunctor1C** (p. 330). The return is a boolean, while the functor takes an **ArRobotPacket** (p. 406) pointer as the argument. This functor is placed in the list of functors to call when a packet arrives. This list is gone through by order until one of the handlers returns true. @argument functor the functor to call when the packet comes in @argument position whether to place the functor first or last

See also:

remPacketHandler (p. 394)

4.101.4.8 void ArRobot::addRunExitCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when the run loop exits for what ever reason.

Adds a callback that is called when the run loop exits. The functor is which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). The whole list of functors is called when the run loop exits. This is most usefull for threaded programs that run the robot using **ArRobot::runAsync** (p. 397). This will allow user threads to know when the robot loop has exited.

Parameters:

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remRunExitCB (p. 395)

4.101.4.9 bool ArRobot::addSensorInterpTask (const char * name, int position, ArFunctor * functor, ArTaskState::State * state = NULL)

Adds a task under the sensor interp part of the syncronous tasks.

The synchronous tasks get called every robot cycle (every 100 ms by default).

Parameters:

name the name to give to the task, should be unique

position the place in the list of user tasks to place this task, this can be any integer, though by convention 0 to 100 is used. The tasks are called in order of highest number to lowest number.

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

See also:

remSensorInterpTask (p. 395)

4.101.4.10 bool ArRobot::addUserTask (const char * name, int position, ArFunctor * functor, ArTaskState::State * state = NULL)

Adds a user task to the list of synchronous taskes.

The synchronous tasks get called every robot cycle (every 100 ms by default).

Parameters:

name the name to give to the task, should be unique

position the place in the list of user tasks to place this task, this can be any integer, though by convention 0 to 100 is used. The tasks are called in order of highest number to lowest position number.

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

See also:

remUserTask (p. 396)

4.101.4.11 void ArRobot::applyTransform (ArTransform trans, bool doCumulative = true)

This applies a transform to all the robot range devices and to the sonar.

Applies a transform to the range devices... this is mostly useful for translating to/from local/global coords, but may have other uses

Parameters:

trans the transform to apply

doCumulative whether to transform the cumulative buffers or not

4.101.4.12 bool ArRobot::asyncConnect (void)

Connects to a robot, from the robots own thread.

Sets up the robot to connect, then returns, but the robot must be running (ie from runAsync) before you do this. Also this will fail if the robot is already connected. If you want to know what happened because of the connect then look at the callbacks. NOTE, this will not lock robot before setting values, so you MUST lock the robot before you call this function and unlock the robot after you call this function. If you fail to lock the robot, you'll may wind up with wierd behavior. Other than the aspect of blocking or not the only difference between async and blocking connects (other than the blocking) is that async is run every robot cycle, whereas blocking runs as fast as it can... also blocking will try to reconnect a radio modem if it looks like it didn't get connected in the first place, so blocking can wind up taking 10 or 12 seconds to decide it can't connect, whereas async doesn't try hard at all to reconnect the radio modem (beyond its first try) (under the assumption the async connect is user driven, so they'll just try again, and so that it won't mess up the sync loop by blocking for so long).

Returns:

true if the robot is running and the robot will try to connect, false if the robot isn't running so won't try to connect or if the robot is already connected

```
See also:
addConnectCB (p. 372)
See also:
addFailedConnectCB (p. 373)
See also:
runAsync (p. 397)
```

4.101.4.13 int ArRobot::asyncConnectHandler (bool tryHarderToConnect)

Internal function, shouldn't be used, does a single run of connecting.

This is an internal function that is used both for async connects and blocking connects use to connect. It does about the same thing for both, and it should only be used by asyncConnect and blockingConnect really. But here it is. The only difference between when its being used by blocking/async connect is that in blocking mode if it thinks there may be problems with the radio modem it pauses for two seconds trying to deal with this... whereas in async mode it tries to deal with this in a simpler way.

Parameters:

tryHarderToConnect if this is true, then if the radio modems look like they aren't working, it'll take about 2 seconds to try and connect them, whereas if its false, it'll do a little try, but won't try very hard

Returns:

0 if its still trying to connect, 1 if it connected, 2 if it failed

4.101.4.14 void ArRobot::attachKeyHandler (ArKeyHandler * keyHandler, bool exitOnEscape = true)

Attachs a key handler.

This will attach a key handler to a robot, by putting it into the robots sensor interp task list (a keyboards a sensor of users will, right?). By default exitOn-Escape is true, which will cause this function to add an escape key handler to the key handler, this will make the program exit when escape is pressed... if you don't like this you can pass exitOnEscape in as false.

Parameters:

keyHandler the key handler to attachexitOnEscape whether to exit when escape is pressed or not

4.101.4.15 bool ArRobot::blockingConnect (void)

Connects to a robot, not returning until connection made or failed.

Connects to the robot, returning only when a connection has been made or it has been established a connection can't be made. This connection usually is fast, but can take up to 30 seconds if the robot is in a wierd state (this is not often). If the robot is connected via ArSerialConnection (p. 424) then the connect will also connect the radio modems. Upon a successful connection all of the Connection Callback Functors that have been registered will be called. NOTE, this will lock the robot before setting values, so you MUST not have the robot locked from where you call this function. If you do, you'll wind up in a deadlock. This behavior is there because otherwise you'd have to lock the robot before calling this function, and normally blockingConnect will be called from a seperate thread, and that thread won't be doing anything else with the robot at that time. Other than the aspect of blocking or not the only difference between async and blocking connects (other than the blocking) is that async is run every robot cycle, whereas blocking runs as fast as it can... also blocking will try to reconnect a radio modem if it looks like it didn't get connected in the first place, so blocking can wind up taking 10 or 12 seconds to decide it can't connect, whereas async doesn't try hard at all to reconnect the radio modem (under the assumption the async connect is user driven, so they'll just try again, and so that it won't mess up the sync loop by blocking for so long).

Returns:

true if a connection could be made, false otherwise

4.101.4.16 double ArRobot::checkRangeDevicesCumulativeBox (double x1, double y1, double x2, double y2, ArPose * readingPos = NULL) const

This goes through all of the registered range devices and locks each, calls cumulativeReadingBox on it, and then unlocks it.

Gets the closest reading in a region defined by the two points of a rectangle.

Parameters:

x1 the x coordinate of one of the rectangle points

y1 the v coordinate of one of the rectangle points

x2 the x coordinate of the other rectangle point

y2 the y coordinate of the other rectangle point

readingPos a pointer to a position in which to store the location of the closest position

Returns:

if the return is >= 0 then this is the distance to the closest reading, if it is < 0 then there were no readings in the given region

4.101.4.17 double ArRobot::checkRangeDevicesCumulativePolar (double startAngle, double endAngle, double * angle = NULL) const

Goes through all the range devices and checks them.

This goes through all of the registered range devices and locks each, calls cumulativeReadingPolar on it, and then unlocks it.

Gets the closest reading in a region defined by startAngle going to endAngle... going counterclockwise (neg degrees to poseitive... with how the robot is set up, thats counterclockwise)... from -180 to 180... this means if you want the slice between 0 and 10 degrees, you must enter it as 0, 10, if you do 10, 0 you'll get the 350 degrees between 10 and 0... be especially careful with negative... for example -30 to -60 is everything from -30, around through 0, 90, and 180 back to -60... since -60 is actually to clockwise of -30

Parameters:

startAngle where to start the sliceendAngle where to end the slice, going clockwise from startAngleangle a pointer return of the angle to the found reading

Returns:

if the return is >= 0 then this is the distance to the closest reading, if it is < 0 then there were no readings in the given region

4.101.4.18 double ArRobot::checkRangeDevicesCurrentBox (double x1, double y1, double x2, double y2, ArPose * readingPos = NULL) const

This goes through all of the registered range devices and locks each, calls current-ReadingBox on it, and then unlocks it.

Gets the closest reading in a region defined by the two points of a rectangle.

Parameters:

x1 the x coordinate of one of the rectangle points

y1 the y coordinate of one of the rectangle points

x2 the x coordinate of the other rectangle point

y2 the y coordinate of the other rectangle point

readingPos a pointer to a position in which to store the location of the closest position

Returns:

if the return is >= 0 then this is the distance to the closest reading, if it is < 0 then there were no readings in the given region

4.101.4.19 double ArRobot::checkRangeDevicesCurrentPolar (double startAngle, double endAngle, double * angle = NULL) const

Goes through all the range devices and checks them.

This goes through all of the registered range devices and locks each, calls current-ReadingPolar on it, and then unlocks it.

Gets the closest reading in a region defined by startAngle going to endAngle... going counterclockwise (neg degrees to poseitive... with how the robot is set up, thats counterclockwise)... from -180 to 180... this means if you want the slice between 0 and 10 degrees, you must enter it as 0, 10, if you do 10, 0 you'll get

the 350 degrees between 10 and 0... be especially careful with negative... for example -30 to -60 is everything from -30, around through 0, 90, and 180 back to -60... since -60 is actually to clockwise of -30

Parameters:

startAngle where to start the sliceendAngle where to end the slice, going clockwise from startAngleangle a pointer return of the angle to the found reading

Returns:

if the return is >= 0 then this is the distance to the closest reading, if it is < 0 then there were no readings in the given region

4.101.4.20 void ArRobot::clearDirectMotion (void)

Clears what direct motion commands have been given, so actions work.

This clears the direct motion commands so that actions will be allowed to control the robot again.

See also:

 $\begin{array}{lll} \textbf{setDirectMotionPrecedenceTime} & (p.\,399) &, & \textbf{getDirectMotion-PrecedenceTime} & (p.\,386) \\ \end{array}$

4.101.4.21 bool ArRobot::com (unsigned char command)

Sends a command to the robot with no arguments.

Parameters:

command the command number to send

Returns

whether the command could be sent or not

4.101.4.22 bool ArRobot::com2Bytes (unsigned char command, char high, char low)

Sends a command to the robot with two bytes for argument.

Parameters:

command the command number to send

high the high byte to send with the command low the low byte to send with the command

Returns:

whether the command could be sent or not

4.101.4.23 bool ArRobot::comInt (unsigned char *command*, short int *argument*)

Sends a command to the robot with an int for argument.

Parameters:

command the command number to send
argument the integer argument to send with the command

Returns:

whether the command could be sent or not

4.101.4.24 bool ArRobot::comStr (unsigned char *command*, const char * *argument*)

Sends a command to the robot with a string for argument.

Parameters:

command the command number to send str the string to send with the command

Returns:

whether the command could be sent or not

4.101.4.25 bool ArRobot::comStrN (unsigned char command, const char * str, int size)

Sends a command to the robot with a size bytes of str as argument.

Parameters:

 ${\color{blue} command}$ the command number to send ${\color{blue} str}$ the character array to send with the command ${\color{blue} size}$ length of the array to send

Returns:

whether the command could be sent or not

4.101.4.26 void ArRobot::disableMotors ()

Disables the motors on the robot.

This command disables the motors on the robot, if it is connected.

4.101.4.27 bool ArRobot::disconnect (void)

Disconnects from a robot.

Disconnects from a robot. This also calls of the DisconnectNormally Callback Functors if the robot was actually connected to a robot when this member was called.

Returns:

true if not connected to a robot (so no disconnect can happen, but it didn't failed either), also true if the command could be sent to the robot (ie connection hasn't failed)

4.101.4.28 void ArRobot::enableMotors ()

Enables the motors on the robot.

This command enables the motors on the robot, if it is connected.

4.101.4.29 ArAction * ArRobot::findAction (const char * actionName)

Returns the first (highest priority) action with the given name (or NULL).

Finds the action with the given name... if more than one action has that name it find the one with the highest priority

Parameters:

actionName the name of the action we want to find

Returns:

the action, if found. If not found, NULL

4.101.4.30 ArRangeDevice * ArRobot::findRangeDevice (const char * name)

Finds a rangeDevice in the robot's list.

Parameters:

name return the first device with this name

Returns:

if found, a range device with the given name, if not found NULL

4.101.4.31 const ArRangeDevice * ArRobot::findRangeDevice (const char * name) const

Finds a rangeDevice in the robot's list.

Parameters:

name return the first device with this name

Returns:

if found, a range device with the given name, if not found NULL

4.101.4.32 ArSyncTask * ArRobot::findTask (ArFunctor * functor)

Finds a task by functor.

Finds a task by its functor, searching the entire space of tasks

Returns:

NULL if no task with that functor found, otherwise a pointer to the **Ar-SyncTask** (p. 475) for the first task found with that functor

4.101.4.33 ArSyncTask * ArRobot::findTask (const char * name)

Finds a task by name.

Finds a task by its name, searching the entire space of tasks

Returns:

NULL if no task of that name found, otherwise a pointer to the **ArSync-Task** (p. 475) for the first task found with that name

4.101.4.34 ArSyncTask * ArRobot::findUserTask (ArFunctor * functor)

Finds a user task by functor.

Finds a user task by its functor, searching the entire space of tasks

Returns:

NULL if no user task with that functor found, otherwise a pointer to the **ArSyncTask** (p. 475) for the first task found with that functor

4.101.4.35 ArSyncTask * ArRobot::findUserTask (const char * name)

Finds a user task by name.

Finds a user task by its name, searching the entire space of tasks

Returns:

NULL if no user task of that name found, otherwise a pointer to the **Ar-SyncTask** (p. 475) for the first task found with that name

$\begin{array}{ll} \textbf{4.101.4.36} & \textbf{ArResolver::ActionMap} * \textbf{ArRobot::getActionMap} \\ & (\textbf{void}) \end{array}$

Returns the map of actions... don't do this unless you really know what you're doing.

This returns the actionMap the robot has... do not mess with this list except by using **ArRobot::addAction** (p. 371) and **ArRobot::remAction** (p. 393)... This is jsut for the things like **ArActionGroup** (p. 60) that want to deactivate or activate all the actions (well, only deactivating everything makes sense).

Returns:

the actions the robot is using

4.101.4.37 unsigned int ArRobot::getConnectionCycleMultiplier (void) const

Gets the multiplier for how many cycles ArRobot waits when connecting.

Returns:

when the ArRobot is waiting for a connection packet back from a robot, it waits for this multiplier times the cycle time for the packet to come back before it gives up on it... This should be small for normal connections but if doing something over a slow network then you may want to make it larger

4.101.4.38 int ArRobot::getConnectionTimeoutTime (void) const

Gets the time without a response until connection assumed lost.

Gets the number of seconds to go without response from the robot until it is assumed that the connection with the robot has been broken and the disconnect on error events will happen.

4.101.4.39 double ArRobot::getControl (void) const [inline]

Gets the control heading.

Gets the control heading as an offset from the current heading.

See also:

getTh (p. 358)

4.101.4.40 unsigned int ArRobot::getCycleTime (void) const

Gets the number of ms between cycles.

Finds the number of milliseconds between cycles, at each cycle is when all packets are processed, all sensors are interpretted, all actions are called, and all user tasks are serviced. Be warned, if you set this too small you could overflow your serial connection.

Returns:

the number of milliseconds between cycles

$\begin{array}{ll} \textbf{4.101.4.41} & \textbf{ArDeviceConnection} * \textbf{ArRobot::getDeviceConnection} \\ & (\textbf{void}) \ \textbf{const} \end{array}$

Gets the connection this instance uses.

Gets the connection this instance uses to the actual robot. This is where commands will be sent and packets will be received from

Returns:

the deviceConnection used for this robot

See also:

 $\begin{aligned} \mathbf{ArDeviceConnection} \ (p.\ 124) \ , \ \mathbf{ArSerialConnection} \ (p.\ 424) \ , \ \mathbf{ArTcp-Connection} \ (p.\ 481) \end{aligned}$

4.101.4.42 unsigned int ArRobot::getDirectMotionPrecedenceTime (void) const

Gets the length of time a direct motion command will take precedence over actions, in milliseconds.

The direct motion precedence time determines how long actions will be ignored after a direct motion command is given. If the direct motion precedence time is 0, then direct motion will take precedence over actions until a clearDirectMotion command is issued. This value defaults to 0.

Returns

the number of milliseconds direct movement will trump actions

See also:

 $\begin{array}{lll} \textbf{setDirectMotionPrecedenceTime} & (p.\,399) &, & \textbf{clearDirectMotion} \\ (p.\,380) & \end{array}$

4.101.4.43 ArRetFunctor1< double, ArPoseWithTime > * ArRobot::getEncoderCorrectionCallback (void) const

Gets the encoderCorrectionCallback.

This gets the encoderCorrectionCB, see setEncoderCorrectionCallback for details.

Returns:

the callback, or NULL if there isn't one

4.101.4.44 ArTransform ArRobot::getEncoderTransform (void) const

Gets the encoder transform.

Returns:

the transform from encoder to global coords

4.101.4.45 ArTime ArRobot::getLastPacketTime (void) const

Gets the time the last packet was received.

This gets the **ArTime** (p. 491) that the last packet was received at

Returns

the time the last packet was received

4.101.4.46 int ArRobot::getPoseInterpPosition (ArTime timeStamp, ArPose * position) [inline]

Gets the position the robot was at at the given timestamp.

See also:

ArInterpolation::getPose (p. 211)

4.101.4.47 std::list< ArRangeDevice *> * ArRobot::getRangeDeviceList (void)

Gets the range device list.

This gets the list of range devices attached to this robot, do NOT manipulate this list directly. If you want to manipulate use the appropriate addRange-Device, or remRangeDevice

Returns:

the list of range dvices attached to this robot

$\begin{array}{ll} \textbf{4.101.4.48} & const \ ArRobotParams * ArRobot::getRobotParams \\ & (void) \ const \end{array}$

Gets the parameters the robot is using.

Returns:

the ArRobotParams (p. 415) instance the robot is using for its parameters

4.101.4.49 int ArRobot::getSonarRange (int num) const

Gets the range of the last sonar reading for the given sonar.

Parameters:

num the sonar number to check, should be between 0 and the number of sonar, the function won't fail if a bad number is given, will just return -1

Returns:

-1 if the sonar has never returned a reading, otherwise the sonar range, which is the distance from the physical sonar disc to where the sonar bounced back

See also:

getNumSonar (p. 361)

4.101.4.50 ArSensorReading * ArRobot::getSonarReading (int num) const

Returns the sonar reading for the given sonar.

Parameters:

num the sonar number to check, should be between 0 and the number of sonar, the function won't fail if a bad number is given, will just return false

Returns:

NULL if there is no sonar defined for the given number, otherwise it returns a pointer to an instance of the **ArSensorReading** (p. 419), note that this class retains ownership, so the instance pointed to should not be deleted and no pointers to it should be stored. Note that often there will be sonar defined but no readings for it (since the readings may be created by the parameter reader), if there has never been a reading from the sonar then the range of that sonar will be -1 and its counterTaken value will be 0

4.101.4.51 int ArRobot::getStateReflectionRefreshTime (void) const

Sets the number of milliseconds between state reflection refreshes if the state has not changed.

The state reflection refresh time is the number of milliseconds between when the state reflector will refresh the robot, if the command hasn't changed. The default is 500 milliseconds. If this number is less than the cyle time, it'll simply happen every cycle.

Returns:

the state reflection refresh time

4.101.4.52 ArSyncTask * ArRobot::getSyncTaskRoot (void)

This gets the root of the syncronous task tree, only serious developers should use it.

This gets the root of the synchronous task tree, so that someone can add their own new types of tasks, or find out more information about each task... only serious developers should use this.

Returns:

the root of the sycnhronous task tree

See also:

ArSyncTask (p. 475)

4.101.4.53 ArTransform ArRobot::getToGlobalTransform (void) const

This gets the transform from local coords to global coords.

Returns:

an **ArTransform** (p. 493) which can be used for transforming a position in local coordinates to one in global coordinates

4.101.4.54 ArTransform ArRobot::getToLocalTransform (void) const

This gets the transform for going from global coords to local coords.

Returns

an **ArTransform** (p. 493) which can be used for transforming a position in global coordinates to one in local coordinates

4.101.4.55 bool ArRobot::hasRangeDevice (ArRangeDevice * device) const

Finds whether a particular range device is attached to this robot or not.

Parameters:

device the device to check for

4.101.4.56 void ArRobot::init (void)

Internal function, shouldn't be used.

Sets up the packet handlers, sets up the sync list and makes the default priority resolver.

4.101.4.57 bool ArRobot::isConnected (void) const [inline]

Questions whether the robot is connected or not.

Returns

true if connected to a robot, false if not

4.101.4.58 bool ArRobot::isDirectMotion (void) const

Returns true if direct motion commands are blocking actions.

Returns the state of direct motion commands: whether actions are allowed or not

See also:

clearDirectMotion (p. 380)

4.101.4.59 bool ArRobot::isHeadingDone (double delta = 0.0) const

Sees if the robot is done changing to the previously given setHeading.

Determines if a setHeading command is finished, to within a small distance. If delta = 0 (default), the delta distance is what was set with setHeadingDoneDiff, you can get the distance with getHeadingDoneDiff

Parameters:

delta how close to the goal distance the robot must be

Returns:

true if the robot has achieved the heading given in a move command or if the robot is no longer in heading mode mode (because its now running off of actions, setDHeading, or setRotVel was called).

4.101.4.60 bool ArRobot::isMoveDone (double delta = 0.0)

Sees if the robot is done moving the previously given move.

Determines if a move command is finished, to within a small distance. If delta = 0 (default), the delta distance is what was set with setMoveDoneDist, you can get the distance with getMoveDoneDist

Parameters:

delta how close to the goal distance the robot must be

Returns:

true if the robot has finished the distance given in a move command or if the robot is no longer in a move mode (because its now running off of actions, setVel, or setVel2 was called).

4.101.4.61 bool ArRobot::isRunning (void) const

Returns whether the robot is currently running or not.

Returns:

true if the robot is currently running in a run or runAsync, otherwise false

4.101.4.62 bool ArRobot::isSonarNew (int num) const

Find out if the given sonar has a new reading.

Parameters:

num the sonar number to check, should be between 0 and the number of sonar, the function won't fail if a bad number is given, will just return false

Returns:

false if the sonar reading is old, or if there is no reading from that sonar

4.101.4.63 bool ArRobot::loadParamFile (const char * file)

Loads a parameter file (replacing all other params).

Returns:

true if the file could be loaded, false otherwise

4.101.4.64 void ArRobot::logAllTasks (void) const

Logs the list of all tasks, strictly for your viewing pleasure.

See also:

ArLog (p. 231)

4.101.4.65 void ArRobot::logUserTasks (void) const

Logs the list of user tasks, strictly for your viewing pleasure.

See also:

ArLog (p. 231)

4.101.4.66 void ArRobot::loopOnce (void)

This function loops once... only serious developers should use it.

This function is only for serious developers, it basically runs the loop once. You would use this function if you were wanting to use robot control in some other monolithic program, so you could work within its framework, rather than trying to get it to work in ARIA.

4.101.4.67 void ArRobot::move (double distance)

Move the given distance forward/backwards

See also:

clearDirectMotion (p. 380).

Tells the robot to move the specified distance forward/backwards, if the constructor was created with state reflecting enabled then it caches this value, and sends it during the next cycle. If state reflecting is disabled it sends this value instantly.

Parameters:

distance the distance for the robot to move

4.101.4.68 void ArRobot::moveTo (ArPose poseTo, ArPose poseFrom, bool doCumulative = true)

Moves the robot's RW position to reflect pose From => pose To.

Parameters:

pose To the absolute real world position to move topose From the original absolute real world positiondo Cumulative whether to update the cumulative buffers or not

4.101.4.69 void ArRobot::moveTo (ArPose pose, bool doCumulative = true)

Moves the robot's idea of its position to this position.

Parameters:

pose the absolute real world position to place the robotdoCumulative whether to update the cumulative buffers or not

4.101.4.70 void ArRobot::packetHandler (void)

Packet Handler, internal.

Reads in all of the packets that are available to read in, then runs through the list of packet handlers and tries to get each packet handled.

See also:

addPacketHandler (p. 374), remPacketHandler (p. 394)

4.101.4.71 bool ArRobot::remAction (const char * actionName)

Removes an action from the list, by name.

Finds the action with the given name and removes it from the actions... if more than one action has that name it find the one with the lowest priority

Parameters:

actionName the name of the action we want to find

Returns:

whether remAction found anything with that action to remove or not

4.101.4.72 bool ArRobot::remAction (ArAction * action)

Removes an action from the list, by pointer.

Finds the action with the given pointer and removes it from the actions... if more than one action has that pointer it find the one with the lowest priority

Parameters:

action the action we want to remove

Returns:

whether remAction found anything with that action to remove or not

4.101.4.73 void ArRobot::remConnectCB (ArFunctor * functor)

Adds a disconnect callback.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addConnectCB (p. 372)

4.101.4.74 void ArRobot::remDisconnectNormallyCB (ArFunctor * functor)

Removes a callback for when disconnect is called while connected.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addDisconnectNormallyCB (p. 372)

4.101.4.75 void ArRobot::remDisconnectOnErrorCB (ArFunctor * functor)

Removes a callback for when disconnection happens because of an error.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addDisconnectOnErrorCB (p. 373)

4.101.4.76 void ArRobot::remFailedConnectCB (ArFunctor * functor)

Removes a callback for when a connection to the robot is failed.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addFailedConnectCB (p. 373)

4.101.4.77 void ArRobot::remPacketHandler (ArRetFunctor1 < bool, ArRobotPacket *> * functor)

Removes a packet handler from the list of packet handlers.

Parameters:

functor the functor to remove from the list of packet handlers

See also:

addPacketHandler (p. 374)

4.101.4.78 void ArRobot::remRangeDevice (ArRangeDevice * device)

Remove a range device from the robot's list, by instance.

Parameters:

device remove the first device with this pointer value

4.101.4.79 void ArRobot::remRangeDevice (const char * name)

Remove a range device from the robot's list, by name.

Parameters:

name remove the first device with this name

4.101.4.80 void ArRobot::remRunExitCB (ArFunctor * functor)

Removes a callback for when the run loop exits for what ever reason.

Parameters:

functor the functor to remove from the list of run exit callbacks

See also:

addRunExitCB (p. 374)

4.101.4.81 void ArRobot::remSensorInterpTask (ArFunctor * functor)

Removes a sensor interp tasks by functor.

See also:

addSensorInterpTask (p. 374), remSensorInterpTask(std::string name)

4.101.4.82 void ArRobot::remSensorInterpTask (const char * name)

Removes a sensor interp tasks by name.

See also:

```
 \begin{array}{lll} \textbf{addSensorInterpTask} & (p.\,374) &, & \textbf{remSensorInterpTask} (ArFunctor *functor) & (p.\,395) \\ \end{array}
```

4.101.4.83 void ArRobot::remUserTask (ArFunctor * functor)

Removes a user task from the list of synchronous taskes by functor.

See also:

addUserTask (p. 375), remUserTask(std::string name)

4.101.4.84 void ArRobot::remUserTask (const char * name)

Removes a user task from the list of synchronous taskes by name.

See also:

addUserTask (p. 375), remUserTask(ArFunctor *functor) (p. 396)

4.101.4.85 void ArRobot::robotLocker (void)

Robot locker, internal.

This just locks the robot, so that its locked for all the user tasks

4.101.4.86 void ArRobot::robotUnlocker (void)

Robot unlocker, internal.

This just unlocks the robot

4.101.4.87 void ArRobot::run (bool stopRunIfNotConnected)

Starts the instance to do processing in this thread.

This starts the list of tasks to be run through until stopped. This function doesn't return until something calls stop on this instance.

Parameters:

stopRunIfNotConnected if true, the run will return if there is no connection to the robot at any given point, this is good for one-shot programs... if it is false the run won't return unless stop is called on the instance

4.101.4.88 void ArRobot::runAsync (bool stopRunIfNotConnected)

Starts the instance to do processing in its own new thread.

This starts a new thread then has runs through the tasks until stopped. This function doesn't return until something calls stop on this instance. This function returns immediately

Parameters:

stopRunIfNotConnected if true, the run will stop if there is no connection to the robot at any given point, this is good for one-shot programs... if it is false the run won't stop unless stop is called on the instance

4.101.4.89 void ArRobot::setConnectionCycleMultiplier (unsigned int multiplier)

Sets the multiplier for how many cycles ArRobot waits when connecting.

Parameters:

multiplier when the ArRobot is waiting for a connection packet back from a robot, it waits for this multiplier times the cycle time for the packet to come back before it gives up on it... This should be small for normal connections but if doing something over a slow network then you may want to make it larger

4.101.4.90 void ArRobot::setConnectionTimeoutTime (int mSecs)

Sets the time without a response until connection assumed lost.

Sets the number of seconds to go without a response from the robot until it is assumed that the connection with the robot has been broken and the disconnect on error events will happen. Note that this will only happen with the default packet handler.

Parameters:

seconds if seconds is 0 then the connection timeout feature will be disabled, otherwise disconnect on error will be triggered after this number of seconds...

4.101.4.91 void ArRobot::setCycleTime (unsigned int ms)

Sets the number of ms between cycles.

Sets the number of milliseconds between cycles, at each cycle is when all packets are processed, all sensors are interpretted, all actions are called, and all user tasks are serviced. Be warned, if you set this too small you could overflow your serial connection.

Parameters:

ms the number of milliseconds between cycles

4.101.4.92 void ArRobot::setDeadReconPose (ArPose pose)

Sets the dead recon position of the robot.

Parameters:

pose the position to set the dead recon position to

4.101.4.93 void ArRobot::setDeltaHeading (double deltaHeading)

Sets the delta heading

See also:

clearDirectMotion (p. 380).

Sets a delta heading to the robot, if the constructor was created with state reflecting enabled then it caches this value, and sends it during the next cycle. If state reflecting is disabled it sends this value instantly.

Parameters:

deltaHeading the desired amount to change the heading of the robot by

4.101.4.94 void ArRobot::setDeviceConnection (ArDeviceConnection * connection)

Sets the connection this instance uses.

Sets the connection this instance uses to the actual robot. This is where commands will be sent and packets will be received from

Parameters:

connection The deviceConnection to use for this robot

See also:

ArDeviceConnection (p. 124), ArSerialConnection (p. 424), ArTcp-Connection (p. 481)

4.101.4.95 void ArRobot::setDirectMotionPrecedenceTime (int mSec)

Sets the length of time a direct motion command will take precedence over actions, in milliseconds.

The direct motion precedence time determines how long actions will be ignored after a direct motion command is given. If the direct motion precedence time is 0, then direct motion will take precedence over actions until a clearDirectMotion command is issued. This value defaults to 0.

Parameters:

the number of milliseconds direct movement should trump actions, if a negative number is given, then the value will be 0

See also:

setDirectMotionPrecedenceTime (p. 399) , clearDirectMotion (p. 380)

4.101.4.96 void ArRobot::setEncoderCorrectionCallback (ArRetFunctor1< double, ArPoseWithTime >* functor)

Sets the encoderCorrectionCallback.

This sets the encoderCorrectionCB, this callback returns the robots change in heading, it takes in the change in heading, x, and y, between the previous and current readings.

Parameters:

functor an ArRetFunctor1 (p. 328) created as an ArRetFunctor1C (p. 330), that will be the callback... call this function NULL to clear the callback

See also:

getEncoderCorrectionCallback (p. 386)

4.101.4.97 void ArRobot::setEncoderTransform (ArPose transformPos)

Changes the transform directly.

Parameters:

transformPos the position to transform to

4.101.4.98 void ArRobot::setEncoderTransform (ArPose deadReconPos, ArPose globalPos)

Changes the transform.

Parameters:

deadReconPos the dead recon position to transform from realWorldPos the real world global position to transform to

4.101.4.99 void ArRobot::setHeading (double heading)

Sets the heading

See also:

clearDirectMotion (p. 380).

Sets the heading of the robot, if the constructor was created with state reflecting enabled then it caches this value, and sends it during the next cycle. If state reflecting is disabled it sends this value instantly.

Parameters:

heading the desired heading of the robot

4.101.4.100 bool ArRobot::setMaxRotVel (double maxVel)

Sets the robots maximum rotational velocity.

This sets the maximum velocity the robot will go... the maximum velocity can also be set by the actions, but it will not be allowed to go higher than this value.

Parameters:

maxVel the maximum velocity to be set, it must be a non-zero number

Returns:

true if the value is good, false othrewise

4.101.4.101 bool ArRobot::setMaxTransVel (double maxVel)

Sets the robots maximum translational velocity.

This sets the maximum velocity the robot will go... the maximum velocity can also be set by the actions, but it will not be allowed to go higher than this value.

Parameters:

maxVel the maximum velocity to be set, it must be a non-zero number

Returns:

true if the value is good, false othrewise

4.101.4.102 void ArRobot::setRotVel (double velocity)

Sets the rotational velocity

See also:

clearDirectMotion (p. 380).

Sets the rotational velocity of the robot, if the constructor was created with state reflecting enabled then it caches this value, and sends it during the next cycle. If state reflecting is disabled it sends this value instantly.

Parameters:

velocity the desired rotational velocity of the robot

4.101.4.103 void ArRobot::setStateReflectionRefreshTime (int mSec)

Sets the number of milliseconds between state reflection refreshes if the state has not changed.

The state reflection refresh time is the number of milliseconds between when the state reflector will refresh the robot, if the command hasn't changed. The default is 500 milliseconds. If this number is less than the cyle time, it'll simply happen every cycle.

Parameters:

mSec the refresh time, in milliseconds, non-negative, if negative is given, then the value will be 0

4.101.4.104 void ArRobot::setVel (double velocity)

Sets the velocity

See also:

clearDirectMotion (p. 380).

Sets the velocity of the robot, if the constructor was created with state reflecting enabled then it caches this value, and sends it during the next cycle. If state reflecting is disabled it sends this value instantly.

Parameters:

velocity the desired translational velocity of the robot

4.101.4.105 void ArRobot::setVel2 (double leftVelocity, double rightVelocity)

Sets the velocity of the wheels independently

See also:

clearDirectMotion (p. 380).

Sets the velocity of each of the wheels on the robot independently. if the constructor was created with state reflecting enabled then it caches this value, and sends it during the next cycle. If state reflecting is disabled it sends this value instantly. Note that this cancels both translational velocity AND rotational velocity, and is canceled by any of the other direct motion commands.

Parameters:

leftVelocity the desired velocity of the left wheel
rightVelocity the desired velocity of the right wheel

4.101.4.106 void ArRobot::stateReflector (void)

State Reflector, internal.

If state reflecting (really direct motion command reflecting) was enabled in the constructor (**ArRobot::ArRobot** (p. 371)) then this will see if there are any direct motion commands to send, and if not then send the command given by the actions. If state reflection is disabled this will send a pulse to the robot every state reflection refresh time (setStateReflectionRefreshTime), if you don't wish this to happen simply set this to a very large value.

4.101.4.107 void ArRobot::stop (void)

Stops the robot

See also:

clearDirectMotion (p. 380).

Stops the robot, by telling it to have a translational velocity and rotational velocity of 0. Also note that if you are using actions, this will cause the actions to be ignored until the direct motion precedence timeout has been exceeded or clearDirectMotion is called.

See also:

 $\begin{tabular}{ll} {\bf setDirectMotionPrecedenceTime} & (p. 399) & , & {\bf getDirectMotionPrecedenceTime} & (p. 386) & , & {\bf clearDirectMotion} & (p. 380) \\ \end{tabular}$

4.101.4.108 void ArRobot::stopRunning (bool doDisconnect = true)

Stops the robot from doing any more processing.

This stops this robot from running anymore. If it is stopping from a runAsync it will cause the thread to return (exit), if it is running from a normal run, it will just cause the run function to return.

Parameters:

doDisconnect Disconnect from the robot. Defaulted to true.

4.101.4.109 ArRobot::WaitState ArRobot::waitForConnect (unsigned int msecs = 0)

Suspend calling thread until the ArRobot is connected.

This will suspend the calling thread until the ArRobot's run loop has managed to connect with the robot. There is an optional parameter of milliseconds to wait for the ArRobot to connect. If msecs is set to 0, it will wait until the ArRobot connects. This function will never return if the robot can not be connected with. If you want to be able to handle that case within the calling thread, you must call waitForConnectOrConnFail() (p. 404).

Parameters:

msecs milliseconds in which to wait for the ArRobot to connect

Returns:

WAIT_CONNECTED for success

See also:

 $\begin{tabular}{ll} \bf waitForConnectOrConnFail & (p. 404) & , & \bf wakeAllWaitingThreads \\ (p. 405) & , & \bf wakeAllConnWaitingThreads & (p. 405) & , & \bf wakeAllRunExit-WaitingThreads & (p. 405) \\ \end{tabular}$

4.101.4.110 ArRobot::WaitState ArRobot::waitFor-ConnectOrConnFail (unsigned int msecs = 0)

Suspend calling thread until the ArRobot is connected or fails to connect.

This will suspend the calling thread until the ArRobot's run loop has managed to connect with the robot or fails to connect with the robot. There is an optional parameter of milliseconds to wait for the ArRobot to connect. If msecs is set to 0, it will wait until the ArRobot connects.

Parameters:

msecs milliseconds in which to wait for the ArRobot to connect

Returns:

WAIT_CONNECTED for success

See also:

waitForConnect (p. 403)

4.101.4.111 ArRobot::WaitState ArRobot::waitForRunExit (unsigned int msecs = 0)

Suspend calling thread until the ArRobot run loop has exited.

This will suspend the calling thread until the ArRobot's run loop has exited. There is an optional parameter of milliseconds to wait for the ArRobot run loop to exit. If msecs is set to 0, it will wait until the ArRobot run loop exits.

Parameters:

msecs milliseconds in which to wait for the robot to connect

Returns:

WAIT_RUN_EXIT for success

4.101.4.112 void ArRobot::wakeAllConnOrFailWaitingThreads ()

Wake up all threads waiting for connection or connection failure.

This will wake all the threads waiting for the robot to be connected or waiting for the robot to fail to connect.

See also:

4.101.4.113 void ArRobot::wakeAllConnWaitingThreads ()

Wake up all threads waiting for connection.

This will wake all the threads waiting for the robot to be connected.

See also:

```
\begin{tabular}{ll} \bf wake All Waiting Threads & (p. 405) & , & \bf wake All Run Exit Waiting Threads & (p. 405) & ... \\ \end{tabular}
```

4.101.4.114 void ArRobot::wakeAllRunExitWaitingThreads ()

Wake up all threads waiting for the run loop to exit.

This will wake all the threads waiting for the run loop to exit.

See also:

```
\begin{tabular}{ll} \bf wake All Waiting Threads & (p. 405) & , & \bf wake All Conn Waiting Threads \\ (p. 405) & \\ \end{tabular}
```

4.101.4.115 void ArRobot::wakeAllWaitingThreads ()

Wake up all threads waiting on this robot.

This will wake all the threads waiting for various major state changes in this particular ArRobot. This includes all threads waiting for the robot to be connected and all threads waiting for the run loop to exit.

See also:

```
wakeAllConnWaitingThreads (p.405), wakeAllRunExitWaitingThreads (p.405)
```

The documentation for this class was generated from the following files:

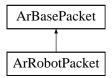
- ArRobot.h
- ArRobot.cpp

4.102 ArRobotPacket Class Reference

Represents the packets sent to the robot as well as those received from it.

#include <ArRobotPacket.h>

Inheritance diagram for ArRobotPacket::



Public Methods

• **ArRobotPacket** (unsigned char sync1=0xfa, unsigned char sync2=0xfb)

Constructor.

• virtual ~ArRobotPacket ()

Destructor.

• bool verifyCheckSum (void)

returns true if the checksum matches what it should be.

• ArTypes::UByte getID (void)

returns the ID of the packet.

• void setID (ArTypes::UByte id)

Sets the ID of the packet.

• ArTypes::Byte2 calcCheckSum (void)

returns the checksum, probably used only internally.

• virtual void finalizePacket (void)

MakeFinals the packet in preparation for sending, must be done.

• ArTime getTimeReceived (void)

Gets the time the packet was received at.

• void **setTimeReceived** (**ArTime** timeReceived)

Sets the time the packet was received at.

4.102.1 Detailed Description

Represents the packets sent to the robot as well as those received from it.

This class reimplements some of the buf operations since the robot is opposeite endian from intel. Also has the getID for convenience.

You can just look at the documentation for the **ArBasePacket** (p. 112) except for the 4 new functions here, verifyCheckSum, getID, print, and calcCheckSum.

4.102.2 Constructor & Destructor Documentation

4.102.2.1 ArRobotPacket::ArRobotPacket (unsigned char sync1 = 0xfa, unsigned char sync2 = 0xfb)

Constructor.

Parameters:

sync1 first byte of the header of this packet, this should be left as the default in nearly all cases, ie don't mess with it

sync2 second byte of the header of this packet, this should be left as the default in nearly all cases, ie don't mess with it

The documentation for this class was generated from the following files:

- ArRobotPacket.h
- ArRobotPacket.cpp

4.103 ArRobotPacketReceiver Class Reference

Given a device connection it receives packets from the robot through it.

#include <ArRobotPacketReceiver.h>

Public Methods

• **ArRobotPacketReceiver** (bool allocatePackets=false, unsigned char sync1=0xfa, unsigned char sync2=0xfb)

Constructor without an already assigned device connection.

• ArRobotPacketReceiver (ArDeviceConnection *deviceConnection, bool allocatePackets=false, unsigned char sync1=0xfa, unsigned char sync2=0xfb)

Constructor with assignment of a device connection.

• virtual ~ArRobotPacketReceiver ()

Destructor.

• ArRobotPacket * receivePacket (unsigned int msWait=0)

Receives a packet from the robot if there is one available.

• void **setDeviceConnection** (**ArDeviceConnection** *device-Connection)

Sets the device this instance receives packets from.

• ArDeviceConnection * getDeviceConnection (void)

Gets the device this instance receives packets from.

• bool isAllocatingPackets (void)

Gets whether or not the receiver is allocating packets.

4.103.1 Detailed Description

Given a device connection it receives packets from the robot through it.

4.103.2 Constructor & Destructor Documentation

4.103.2.1 ArRobotPacketReceiver::ArRobotPacketReceiver (bool allocatePackets = false, unsigned char sync1 = 0xfa, unsigned char sync2 = 0xfb)

Constructor without an already assigned device connection.

Parameters:

allocatePackets whether to allocate memory for the packets before returning them (true) or to just return a pointer to an internal packet (false)... most everything should use false as this will help prevent many memory leaks or corruptions

sync1 first byte of the header this receiver will receive, this should be left as the default in nearly all cases, ie don't mess with it

sync2 second byte of the header this receiver will receive, this should be left as the default in nearly all cases, ie don't mess with it

4.103.2.2 ArRobotPacketReceiver::ArRobotPacketReceiver (ArDeviceConnection * deviceConnection, bool allocatePackets = false, unsigned char sync1 = 0xfa, unsigned char sync2 = 0xfb)

Constructor with assignment of a device connection.

Parameters:

deviceConnection the connection which the receiver will use

allocatePackets whether to allocate memory for the packets before returning them (true) or to just return a pointer to an internal packet (false)... most everything should use false as this will help prevent many memory leaks or corruptions

sync1 first byte of the header this receiver will receive, this should be left as the default in nearly all cases, ie don't mess with it

sync2 second byte of the header this receiver will receive, this should be left as the default in nearly all cases, ie don't mess with it

4.103.3 Member Function Documentation

4.103.3.1 ArRobotPacket * ArRobotPacketReceiver::receivePacket (unsigned int msWait = 0)

Receives a packet from the robot if there is one available.

Parameters:

ms Wait how long to block for the start of a packet, nonblocking if 0

Returns:

NULL if there are no packets in alloted time, otherwise a pointer to the packet received, if allocatePackets is true than the place that called this function owns the packet and should delete the packet when done... if allocatePackets is false then nothing must store a pointer to this packet, the packet must be used and done with by the time this method is called again

The documentation for this class was generated from the following files:

- ArRobotPacketReceiver.h
- ArRobotPacketReceiver.cpp

4.104 ArRobotPacketSender Class Reference

Given a device connection this sends commands through it to the robot.

#include <ArRobotPacketSender.h>

Public Methods

• **ArRobotPacketSender** (unsigned char sync1=0xfa, unsigned char sync2=0xfb)

Constructor without an already assigned device connection.

• ArRobotPacketSender (ArDeviceConnection *deviceConnection, unsigned char sync1=0xfa, unsigned char sync2=0xfb)

Constructor with assignment of a device connection.

• virtual ~ArRobotPacketSender ()

Destructor.

• bool **com** (unsigned char command)

Sends a command to the robot with no arguments.

• bool **comInt** (unsigned char command, short int argument)

Sends a command to the robot with an int for argument.

• bool com2Bytes (unsigned char command, char high, char low)

Sends a command to the robot with two bytes for argument.

• bool comStr (unsigned char command, const char *argument)

Sends a command to the robot with a string for argument.

• bool comStrN (unsigned char command, const char *str, int size)

Sends a command to the robot with a size bytes of str as argument.

• void **setDeviceConnection** (**ArDeviceConnection** *device-Connection)

Sets the device this instance sends commands to.

• ArDeviceConnection * getDeviceConnection (void)

Gets the device this instance sends commands to.

4.104.1 Detailed Description

Given a device connection this sends commands through it to the robot.

4.104.2 Constructor & Destructor Documentation

4.104.2.1 ArRobotPacketSender::ArRobotPacketSender (unsigned char sync1 = 0xfa, unsigned char sync2 = 0xfb)

Constructor without an already assigned device connection.

Parameters:

sync1 first byte of the header this sender will send, this should be left as the default in nearly all cases, ie don't mess with it

sync2 second byte of the header this sender will send, this should be left as the default in nearly all cases, ie don't mess with it

4.104.2.2 ArRobotPacketSender:::ArRobotPacketSender (ArDeviceConnection * deviceConnection, unsigned char sync1 = 0xfa, unsigned char sync2 = 0xfb)

Constructor with assignment of a device connection.

Parameters:

sync1 first byte of the header this sender will send, this should be left as the default in nearly all cases, ie don't mess with it

sync2 second byte of the header this sender will send, this should be left as the default in nearly all cases, ie don't mess with it

4.104.3 Member Function Documentation

4.104.3.1 bool ArRobotPacketSender::com (unsigned char number)

Sends a command to the robot with no arguments.

Parameters:

command the command number to send

Returns:

whether the command could be sent or not

4.104.3.2 bool ArRobotPacketSender::com2Bytes (unsigned char command, char high, char low)

Sends a command to the robot with two bytes for argument.

Parameters:

command the command number to sendhigh the high byte to send with the commandlow the low byte to send with the command

Returns:

whether the command could be sent or not

4.104.3.3 bool ArRobotPacketSender::comInt (unsigned char command, short int argument)

Sends a command to the robot with an int for argument.

Parameters:

command the command number to send
argument the integer argument to send with the command

Returns:

whether the command could be sent or not

4.104.3.4 bool ArRobotPacketSender::comStr (unsigned char command, const char * argument)

Sends a command to the robot with a string for argument.

Parameters:

command the command number to send str the string to send with the command

Returns:

whether the command could be sent or not

4.104.3.5 bool ArRobotPacketSender::comStrN (unsigned char command, const char *str, int size)

Sends a command to the robot with a size bytes of str as argument.

Parameters:

 ${\it command}$ the command number to send ${\it str}$ the character array to send with the command ${\it size}$ length of the array to send

Returns:

whether the command could be sent or not

The documentation for this class was generated from the following files:

- ArRobotPacketSender.h
- ArRobotPacketSender.cpp

4.105 ArRobotParams Class Reference

Contains the robot parameters, according to the parameter file.

#include <ArRobotParams.h>

Public Methods

• ArRobotParams ()

Constructor.

• virtual ~ArRobotParams ()

Destructor.

 $\bullet\,$ void init (ArRobotParamFile *param)

Given the robot parameters in preference form, fills in this instance.

• const char * getClassName (void) const

Returns the class from the parameter file.

• const char * **getSubClassName** (void) const

Returns the subclass from the parameter file.

ullet double $\mathbf{getRobotRadius}$ (void) const

Returns the robot's radius.

• double **getRobotDiagonal** (void) const

Returns the robot diagonal (half-height to diagonal of octagon).

• bool isHolonomic (void) const

Returns whether the robot is holonomic or not.

• bool hasMoveCommand (void) const

Returns if the robot has a built in move command.

• int **getMaxVelocity** (void) const

Returns the max velocity of the robot.

 \bullet int $\mathbf{getMaxRotVelocity}$ (void) const

Returns the max rotational velocity of the robot.

• bool **getRequestIOPackets** (void) const

Returns true if IO packets are automatically requested upon connection to the robot.

• double **getAngleConvFactor** (void) const

Returns the angle conversion factor.

• double **getDistConvFactor** (void) const

Returns the distance conversion factor.

• double **getVelConvFactor** (void) const

Returns the velocity conversion factor.

• double **getRangeConvFactor** (void) const

Returns the sonar range conversion factor.

• double **getDiffConvFactor** (void) const

Returns the wheel velocity difference to angular velocity conv factor.

• double **getVel2Divisor** (void) const

Returns the multiplier for VEL2 commands.

• bool haveTableSensingIR (void) const

Returns true if the robot has table sensing IR.

• bool haveNewTableSensingIR (void) const

Returns true if the robot's table sensing IR bits are sent in the 4th-byte of the IO packet.

• bool haveFrontBumpers (void) const

Returns true if the robot has front bumpers.

• int numFrontBumpers (void) const

Returns the number of front bumpers.

• bool haveRearBumpers (void) const

Returns true if the robot has rear bumpers.

• int numRearBumpers (void) const

Returns the number of rear bumpers.

• int getNumSonar (void) const

Returns the number of sonar.

• bool haveSonar (int number)

Returns if the sonar of the given number is valid.

• int getSonarX (int number)

Returns the X location of the given numbered sonar disc.

• int **getSonarY** (int number)

Returns the Y location of the given numbered sonar disc.

• int **getSonarTh** (int number)

Returns the heading of the given numbered sonar disc.

• bool getLaserPossessed (void) const

Returns if the robot has a laser (according to param file).

• const char * **getLaserPort** (void) const

What port the laser is on.

• bool getLaserPowerControlled (void) const

If the laser power is controlled by the serial port lines.

• bool getLaserFlipped (void) const

If the laser is flipped on the robot.

• int getLaserX (void) const

The X location of the laser.

• int getLaserY (void) const

The Y location of the laser.

4.105.1 Detailed Description

Contains the robot parameters, according to the parameter file.

The documentation for this class was generated from the following files:

- ArRobotParams.h
- ArRobotParams.cpp

4.106 ArSectors Class Reference

A class for keeping track of if a complete revolution has been attained.

#include <ariaUtil.h>

Public Methods

• **ArSectors** (int numSectors=8)

Constructor.

• virtual ~ArSectors ()

Destructor.

• void **clear** (void)

Clears all quadrants.

• void **update** (double angle)

Updates the appropriate quadrant for the given angle.

• bool didAll (void) const

Returns true if the all of the quadrants have been gone through.

4.106.1 Detailed Description

A class for keeping track of if a complete revolution has been attained.

This class can be used to keep track of if a complete revolution has been done, it is used by doing doing a clearQuadrants when you want to stat the revolution. Then at each point doing an updateQuadrant with the current heading of the robot. When didAllQuadrants returns true, then all the quadrants have been done.

The documentation for this class was generated from the following file:

• ariaUtil.h

4.107 ArSensorReading Class Reference

A class to hold a sensor reading, should be one instance per sensor.

#include <ArSensorReading.h>

Public Methods

• **ArSensorReading** (double xPos=0.0, double yPos=0.0, double th-Pos=0.0)

Constructor, the three args are the physical location of the sonar.

• int **getRange** (void)

Gets the range of the reading.

• bool **isNew** (unsigned int counter)

Given the counter from the robot, it returns whether the reading is new.

• double **getX** (void)

Gets the X location of the sensor reading.

• double **getY** (void)

Gets the Y location of the sensor reading.

• ArPose getPose (void)

Gets the position of the reading

Returns:

the position of the reading (ie where the sonar pinged back).

• ArPose getPoseTaken (void)

Gets the pose the reading was taken at.

• ArPose getEncoderPoseTaken (void)

Gets the encoder pose the reading was taken at.

• double **getSensorX** (void)

Gets the X location of the sonar on the robot.

• double **getSensorY** (void)

Gets the Y location of the sensor on the robot.

 \bullet double **getSensorTh** (void)

Gets the heading of the sensor on the robot.

• ArPose getSensorPosition (void)

Gets the sensors position on the robot.

• double **getSensorDX** (void)

Gets the cos component of the heading of the sensor reading.

• double **getSensorDY** (void)

Gets the sin component of the heading of the sensor reading.

• double **getXTaken** (void)

Gets the X locaiton of the robot when the reading was received.

• double **getYTaken** (void)

Gets the Y location of the robot when the reading was received.

• double **getThTaken** (void)

Gets the th (heading) of the robot when the reading was received.

• unsigned int **getCounterTaken** (void)

Gets the counter from when the reading arrived.

• void **newData** (int range, **ArPose** robotPose, **ArPose** encoderPose, **ArTransform** trans, unsigned int counter, **ArTime** timeTaken)

Takes the data and makes the reading reflect it.

• void **ArSensorReading::newData** (int sx, int sy, **ArPose** robotPose, **ArPose** encoderPose, **ArTransform** trans, unsigned int counter, **ArTime** timeTaken)

Takes the data and makes the reading reflect it.

• void **resetSensorPosition** (double xPos, double yPos, double thPos, bool forceComputation=false)

Resets the sensors idea of its physical location on the robot.

• void applyTransform (ArTransform trans)

Applies a transform to the reading position, and where it was taken.

4.107.1 Detailed Description

A class to hold a sensor reading, should be one instance per sensor.

This class holds sensor data and a sensor reading... it can happen that it contains the data for a sonar, but not the reading, in which case the range (from get-Range) will be -1, and the counter it was taken (from getCounterTaken) will be 0, also it will never be new (from isNew)

4.107.2 Constructor & Destructor Documentation

4.107.2.1 ArSensorReading::ArSensorReading (double xPos = 0.0, double yPos = 0.0, double thPos = 0.0)

Constructor, the three args are the physical location of the sonar.

Parameters:

```
xPos the x position of the sensor on the robot (mm) yPos the y position of the sensor on the robot (mm) thPos the heading of the sensor on the robot (deg)
```

4.107.3 Member Function Documentation

4.107.3.1 void ArSensorReading::applyTransform (ArTransform trans)

Applies a transform to the reading position, and where it was taken.

Parameters:

trans the transform to apply to the reading and where the reading was taken

4.107.3.2 unsigned int ArSensorReading::getCounterTaken (void) [inline]

Gets the counter from when the reading arrived.

Returns

the counter from the robot when the sonar reading was taken

See also:

isNew (p. 422)

4.107.3.3 int ArSensorReading::getRange (void) [inline]

Gets the range of the reading.

Returns:

the distance return from the sensor (how far from the robot)

4.107.3.4 ArPose ArSensorReading::getSensorPosition (void) [inline]

Gets the sensors position on the robot.

Returns:

the position of the sensor on the robot

4.107.3.5 bool ArSensorReading::isNew (unsigned int *counter*) [inline]

Given the counter from the robot, it returns whether the reading is new.

Parameters:

counter the counter from the robot at the current time

Returns:

true if the reading was taken on the current loop

See also:

getCounter

4.107.3.6 void ArSensorReading::newData (int range, ArPose robotPose, ArPose encoderPose, ArTransform trans, unsigned int counter, ArTime timeTaken)

Takes the data and makes the reading reflect it.

Parameters:

 ${\it range}$ the distance from the sensor to the sensor return (mm)

x the x location of the robot when the sensor reading was taken (mm)

y the y location of the robot when the sensor reading was taken (mm)

th the heading of the robot when the sensor reading was taken (deg)

trans the transform from local coords to global coords

counter the counter from the robot when the sensor reading was taken

4.107.3.7 void ArSensorReading::resetSensorPosition (double xPos, double yPos, double thPos, bool forceComputation = false)

Resets the sensors idea of its physical location on the robot.

Parameters:

```
xPos the x position of the sensor on the robot (mm) yPos the y position of the sensor on the robot (mm) thPos the heading of the sensor on the robot (deg)
```

The documentation for this class was generated from the following files:

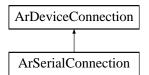
- \bullet Ar
SensorReading.h
- ArSensorReading.cpp

4.108 ArSerialConnection Class Reference

For connecting to devices through a serial port.

#include <ArSerialConnection.h>

Inheritance diagram for ArSerialConnection::



Public Types

 enum Open { OPEN_COULD_NOT_OPEN_PORT = 1, OPEN_-COULD_NOT_SET_UP_PORT, OPEN_INVALID_BAUD_-RATE, OPEN_COULD_NOT_SET_BAUD, OPEN_ALREADY_-OPEN }

Public Methods

• ArSerialConnection ()

Constructor.

• virtual ~ArSerialConnection ()

Destructor also closes the connection.

- int open (const char *port=NULL)
 - $Opens\ the\ serial\ port.$
- void **setPort** (const char *port=NULL)

Sets the port this will use.

• const char * **getPort** (void)

Gets the port this is using.

• virtual bool openSimple (void)

Opens the connection again, using the values from setLocation or.

• virtual int **getStatus** (void)

Gets the status of the connection, which is one of the enum status.

• virtual bool **close** (void)

Closes the connection.

• virtual int **read** (const char *data, unsigned int size, unsigned int ms-Wait=0)

Reads data from connection.

• virtual int write (const char *data, unsigned int size)

Writes data to connection.

• virtual const char * **getOpenMessage** (int messageNumber)

Gets the string of the message associated with opening the device.

• bool **setBaud** (int baud)

Sets the baud rate on the connection.

• int getBaud (void)

Gets what the current baud rate is set to.

• bool **setHardwareControl** (bool hardwareControl)

Sets whether to enable or disable the hardware control lines.

• bool **getHardwareControl** (void)

Gets whether the hardware control lines are enabled or disabled.

• bool **getDCD** (void)

Sees how the DCD is set (true = high).

• bool **getCTS** (void)

Sees how the CTS is set (true = high).

• virtual ArTime getTimeRead (int index)

Gets the time data was read in.

• virtual bool isTimeStamping (void)

sees if timestamping is really going on or not.

4.108.1 Detailed Description

For connecting to devices through a serial port.

4.108.2 Member Enumeration Documentation

4.108.2.1 enum ArSerialConnection::Open

Enumeration values:

OPEN_COULD_NOT_OPEN_PORT Could not open the port.

OPEN_COULD_NOT_SET_UP_PORT Could not set up the port.

OPEN_INVALID_BAUD_RATE Baud rate is not valid.

 $\begin{array}{ll} \mathbf{OPEN_COULD_NOT_SET_BAUD} & \mathrm{Baud\ rate\ valid,\ but\ could\ not\ set} \\ \mathrm{it.} \end{array}$

OPEN_ALREADY_OPEN Connection was already open.

4.108.3 Member Function Documentation

4.108.3.1 bool ArSerialConnection::close (void) [virtual]

Closes the connection.

Returns:

whether the close succeeded or not

Reimplemented from ArDeviceConnection (p. 126).

4.108.3.2 int ArSerialConnection::getBaud (void)

Gets what the current baud rate is set to.

Returns:

the current baud rate of the connection

4.108.3.3 bool ArSerialConnection::getHardwareControl (void)

Gets whether the hardware control lines are enabled or disabled.

Returns:

true if hardware control of lines is enabled, false otherwise

4.108.3.4 const char * ArSerialConnection::getOpenMessage (int messageNumber) [virtual]

Gets the string of the message associated with opening the device.

Each class inherited from this one has an open method which returns 0 for success or an integer which can be passed into this function to obtain a string describing the reason for failure

Parameters:

messageNumber the number returned from the open

Returns:

the error description associated with the messageNumber

Reimplemented from **ArDeviceConnection** (p. 126).

4.108.3.5 const char * ArSerialConnection::getPort (void)

Gets the port this is using.

Returns:

The seiral port to connect to

4.108.3.6 int ArSerialConnection::getStatus (void) [virtual]

Gets the status of the connection, which is one of the enum status.

Gets the status of the connection, which is one of the enum status. If you want to get a string to go along with the number, use getStatusMessage

Returns:

the status of the connection

See also:

getStatusMessage (p. 127)

Reimplemented from **ArDeviceConnection** (p. 126).

4.108.3.7 ArTime ArSerialConnection::getTimeRead (int index) [virtual]

Gets the time data was read in.

Parameters:

index looks like this is the index back in the number of bytes last read in

Returns:

the time the last read data was read in

Reimplemented from **ArDeviceConnection** (p. 127).

4.108.3.8 bool ArSerialConnection::isTimeStamping (void) [virtual]

sees if timestamping is really going on or not.

Returns:

true if real timestamping is happening, false otherwise

Reimplemented from **ArDeviceConnection** (p. 127).

4.108.3.9 int ArSerialConnection::open (const char * port = NULL)

Opens the serial port.

Parameters:

port The serial port to connect to, or NULL which defaults to COM1 for windows and /dev/ttyS0 for linux

Returns:

0 for success, otherwise one of the open enums

See also:

getOpenMessage (p. 427)

4.108.3.10 int ArSerialConnection::read (const char * data, unsigned int size, unsigned int msWait = 0) [virtual]

Reads data from connection.

Reads data from connection

Parameters:

data pointer to a character array to read the data into
size maximum number of bytes to read

msWait read blocks for this many milliseconds (not at all for < 0)

Returns:

number of bytes read, or -1 for failure

See also:

```
write (p. 430), writePacket (p. 129)
```

Reimplemented from ArDeviceConnection (p. 128).

4.108.3.11 bool ArSerialConnection::setBaud (int baud)

Sets the baud rate on the connection.

Parameters:

rate the baud rate to set the connection to

Returns:

whether the set succeeded

See also:

getBaud (p. 426)

4.108.3.12 bool ArSerialConnection::setHardwareControl (bool hardwareControl)

Sets whether to enable or disable the hardware control lines.

Parameters:

hardware Control true to enable hardware control of lines

Returns:

true if the set succeeded

4.108.3.13 void ArSerialConnection::setPort (const char * port = NULL)

Sets the port this will use.

Parameters:

port The serial port to connect to, or NULL which defaults to COM1 for windows and /dev/ttyS0 for linux

See also:

getOpenMessage (p. 427)

4.108.3.14 int ArSerialConnection::write (const char * data, unsigned int size) [virtual]

Writes data to connection.

Writes data to connection

Parameters:

data pointer to a character array to write the data from
size number of bytes to write

Returns:

number of bytes read, or -1 for failure

See also:

read (p. 428), writePacket (p. 129)

Reimplemented from ArDeviceConnection (p. 128).

The documentation for this class was generated from the following files:

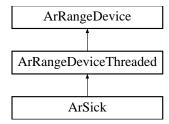
- ArSerialConnection.h
- ArSerialConnection_LIN.cpp
- ArSerialConnection_WIN.cpp

4.109 ArSick Class Reference

The sick driver.

#include <ArSick.h>

Inheritance diagram for ArSick::



Public Types

- enum BaudRate { BAUD9600, BAUD19200, BAUD38400 }
- enum Degrees { DEGREES180, DEGREES100 }
- enum Increment { INCREMENT_ONE, INCREMENT_HALF }

Public Methods

• ArSick (size_t currentBufferSize=361, size_t cumulativeBufferSize=0, const char *name="laser")

Constructor.

• \sim ArSick ()

Destructor.

• void **configure** (bool useSim=false, bool powerControl=true, bool laserFlipped=false, **BaudRate** baud=BAUD38400, **Degrees** deg=DEGREES180, **Increment** incr=INCREMENT_ONE)

Configure the laser before connecting to it.

• void **configureShort** (bool useSim=false, baud=BAUD38400, **Degrees** deg=DEGREES180, **Increment** incr=INCREMENT_ONE)

Shorter configure for the laser.

• void **setSensorPosition** (double x, double y, double th)

Sets the position of the laser on the robot.

ullet void $\mathbf{setSensorPosition}$ (\mathbf{ArPose} pose)

Sets the position of the laser on the robot.

• ArPose getSensorPosition ()

Gets the position of the laser on the robot.

• double **getSensorPositionX** ()

Gets the X position of the laser on the robot.

• double **getSensorPositionY** ()

Gets the Y position of the laser on the robot.

• double **getSensorPositionTh** ()

Gets the heading of the laser on the robot.

• bool **blockingConnect** (void)

Connect to the laser while blocking.

• bool **asyncConnect** (void)

Connect to the laser asyncronously.

• bool **disconnect** (bool doNotLockRobotForSim=false)

Disconnect from the laser.

• void setDeviceConnection (ArDeviceConnection *conn)

Sets the device connection.

• ArDeviceConnection * getDeviceConnection (void)

Gets the device connection.

• bool isConnected (void)

Sees if this is connected to the laser.

• bool **tryingToConnect** (void)

Sees if this is trying to connect to the laser at the moment.

• unsigned int **getMinRange** (void)

Gets the minimum range for this device (defaults to 100 mm).

• void **setMinRange** (unsigned int minRange)

Sets the maximum range for this device (defaults to 100 mm).

• void **setFilterNearDist** (double dist)

Current readings closer than this are discarded as too close.

• double **getFilterNearDist** (void)

Current readings closer than this are discarded as too close.

• void **ArSick::setFilterCumulativeMaxDist** (double dist)

Cumulative readings must be within this distance of the robot to be saved.

• double ArSick::getFilterCumulativeMaxDistance (void)

Cumulative readings must be within this distance of the robot to be saved.

• void ArSick::setFilterCumulativeNearDist (double dist)

Cumulative readings closer than this are discarded as too close.

• double ArSick::getFilterCumulativeNearDistance (void)

Cumulative readings closer than this are discarded as too close.

• void ArSick::setFilterCumulativeCleanDist (double dist)

Cumulative readings that are this close to current beams are discarded.

• double ArSick::getFilterCumulativeCleanDistance (void)

Cumulative readings that are this close to current beams are discarded.

• void ArSick::setFilterCleanCumulativeInterval (int interval)

Cumulative readings are cleaned every this number of milliseconds.

• int ArSick::getFilterCleanCumulativeInterval (void)

Cumulative readings are cleaned every this number of milliseconds.

• bool runOnRobot (void)

Runs the laser off of the robot.

• int getSickPacCount ()

Gets the number of laser packets received in the last second.

void addConnectCB (ArFunctor *functor, ArListPos::Pos position)

Adds a connect callback.

• void remConnectCB (ArFunctor *functor)

Adds a disconnect callback.

• void addFailedConnectCB (ArFunctor *functor, ArListPos::Pos position)

Adds a callback for when a connection to the robot is failed.

• void remFailedConnectCB (ArFunctor *functor)

Removes a callback for when a connection to the robot is failed.

• void addDisconnectNormallyCB (ArFunctor *functor, ArList-Pos::Pos position)

Adds a callback for when disconnect is called while connected.

• void remDisconnectNormallyCB (ArFunctor *functor)

Removes a callback for when disconnect is called while connected.

• void addDisconnectOnErrorCB (ArFunctor *functor, ArList-Pos::Pos position)

Adds a callback for when disconnection happens because of an error.

• void remDisconnectOnErrorCB (ArFunctor *functor)

Removes a callback for when disconnection happens because of an error.

• void **setConnectionTimeoutTime** (int mSecs)

Sets the time without a response until connection assumed lost.

• int getConnectionTimeoutTime (void)

Gets the time without a response until connection assumed lost.

• ArTime getLastReadingTime (void)

Gets the time data was last received.

• bool isUsingSim (void)

Gets whether the laser is simulated or not.

• bool isControllingPower (void)

Gets whether the computer is controling laser power or not.

• bool isLaserFlipped (void)

Gets whether the laser is flipped over or not.

• Degrees getDegrees (void)

Gets the degrees the laser is scanning.

• Increment getIncrement (void)

Gets the amount each scan increments.

• bool simPacketHandler (ArRobotPacket *packet)

The packet handler for when connected to the simulator.

• void sensorInterpCallback (void)

The function called if the laser isn't running in its own thread and isn't simulated.

• bool internalConnectSim (void)

An internal function.

• int internalConnectHandler (void)

An internal function, single loop event to connect to laser.

• virtual void * runThread (void *arg)

The internal function used by the ArRangeDeviceThreaded (p. 320).

void processPacket (ArSickPacket *packet, ArPose pose, ArPose encoderPose, unsigned int counter)

The internal function which processes the sickPackets.

• void runOnce (bool lockRobot)

The internal function that gets does the work.

• virtual void **setRobot** (**ArRobot** *robot)

Sets the robot this device is attached to.

• void **dropConnection** (void)

Internal function, shouldn't be used, drops the conn because of error.

• void **failedConnect** (void)

Internal function, shouldn't be used, denotes the conn failed.

• void madeConnection (void)

Internal function, shouldn't be used, does the after conn stuff.

 \bullet void robotConnectCallback (void)

Internal function, shouldn't be used, gets params from the robot.

Protected Types

enum State { STATE_NONE, STATE_INIT, STATE_WAIT_-FOR_POWER_ON, STATE_CHANGE_BAUD, STATE_CONFIGURE, STATE_WAIT_FOR_CONFIGURE_ACK, STATE_INSTALL_MODE, STATE_WAIT_FOR_INSTALL_MODE_ACK, STATE_SET_MODE, STATE_WAIT_FOR_SET_MODE_ACK, STATE_START_READINGS, STATE_WAIT_FOR_START_ACK, STATE_CONNECTED }

Protected Methods

• void filterReadings ()

Internal function for filtering the raw readings and updating buffers.

- $\bullet\,$ void ${\bf filterAddAndCleanCumulative}$ (double x, double y, bool clean)
 - $Internal\ function\ for\ managing\ the\ cumulative.$
- void filterFarCumulative (void)

Internal function for managing the cumulative.

• void switchState (State state)

Internal function for switching states.

4.109.1 Detailed Description

The sick driver.

4.109.2 Member Enumeration Documentation

4.109.2.1 enum ArSick::BaudRate

Enumeration values:

BAUD9600 9600 Baud.

BAUD19200 19200 Baud.

BAUD38400 38400 Baud.

4.109.2.2 enum ArSick::Degrees

Enumeration values:

DEGREES180 180 Degrees.

DEGREES100 100 Degrees.

4.109.2.3 enum ArSick::Increment

Enumeration values:

INCREMENT_ONE One degree increments.

INCREMENT_HALF Half a degree increments.

4.109.2.4 enum ArSick::State [protected]

Enumeration values:

STATE_NONE Nothing, haven't tried to connect or anything.

STATE_INIT Initializing the laser.

STATE_WAIT_FOR_POWER_ON Waiting for power on.

STATE_CHANGE_BAUD Change the baud, no confirm here.

STATE_CONFIGURE Send the width and increment to the laser.

STATE_WAIT_FOR_CONFIGURE_ACK Wait for the configuration Ack.

STATE_INSTALL_MODE Switch to install mode.

STATE_WAIT_FOR_INSTALL_MODE_ACK Wait until its switched to install mode.

STATE_SET_MODE Set the mode (mm/cm) and extra field bits.

STATE_WAIT_FOR_SET_MODE_ACK Waiting for set-mode ack.

STATE_START_READINGS Switch to monitoring mode.

STATE_WAIT_FOR_START_ACK Waiting for the switch-mode ack.

STATE_CONNECTED We're connected and getting readings.

4.109.3 Member Function Documentation

4.109.3.1 void ArSick::addConnectCB (ArFunctor * functor, ArListPos::Pos position)

Adds a connect callback.

Adds a connect callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). The entire list of connect callbacks is called when a connection is made with the laser. If you have some sort of module that adds a callback, that module must remove the callback when the module is removed.

Parameters:

functorfunctor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remConnectCB (p. 442)

4.109.3.2 void ArSick::addDisconnectNormallyCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when disconnect is called while connected.

Adds a disconnect normally callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). This whole list of disconnect normally callbacks is called when something calls disconnect if the instance is Connected. If there is no connection and disconnect is called nothing is done. If you have some sort of module that adds a callback, that module must remove the callback when the module is removed.

Parameters:

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remFailedConnectCB (p. 443)

4.109.3.3 void ArSick::addDisconnectOnErrorCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when disconnection happens because of an error.

Adds a disconnect on error callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). This whole list of disconnect on error callbacks is called when ARIA loses connection to a laser because of an error. This can occur if the physical connection (ie serial cable) between the laser and the computer is severed/disconnected, or if the laser is turned off. Note that if the link between the two is lost the ARIA assumes it is temporary until it reaches a timeout

value set with setConnectionTimeoutTime. If you have some sort of module that adds a callback, that module must remove the callback when the module removed.

Parameters:

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remFailedConnectCB (p. 443)

4.109.3.4 void ArSick::addFailedConnectCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when a connection to the robot is failed.

Adds a failed connect callback, which is an **ArFunctor** (p. 139), created as an **ArFunctorC** (p. 165). This whole list of failed connect callbacks is called when an attempt is made to connect to the laser, but fails. The usual reason for this failure is either that there is no laser/sim where the connection was tried to be made. If you have some sort of module that adds a callback, that module must remove the callback when the module removed.

Parameters:

functor functor created from ArFunctorC (p. 165) which refers to the function to call.

position whether to place the functor first or last

See also:

remFailedConnectCB (p. 443)

4.109.3.5 bool ArSick::asyncConnect (void)

Connect to the laser asyncronously.

This does not lockDevice the laser, but you should lockDevice the laser before you try to connect. Also note that if you are connecting to the sim the laser MUST be unlocked so that this can lock the laser and send the commands to the sim. To be connected successfully, either the useSim must be set from configure (and the laser must be connected to a simulator, or this will return true but connection will fail), the device must have been run or runasync, or the device must have been runOnLaser.

Returns:

true if a connection will be able to be tried, false otherwise

See also:

configure (p. 440), ArRangeDeviceThreaded::run (p. 320), ArRangeDeviceThreaded::runAsync (p. 320), runOnRobot (p. 443)

4.109.3.6 bool ArSick::blockingConnect (void)

Connect to the laser while blocking.

lockDevice s the laser, and then makes a connection. If it is connecting to the simulator (set with the useSim flag in configure) then it will lock the laser and send the commands to the sim. If where you are calling from has the laser locked, make sure you unlock it before calling this function.

Returns:

true if a connection was made, false otherwise

4.109.3.7 void ArSick::configure (bool useSim = false, bool powerControl = true, bool laserFlipped = false, BaudRate baud = BAUD38400, Degrees deg = DEGREES180, Increment $incr = INCREMENT_ONE$)

Configure the laser before connecting to it.

You must lock Device the laser or not have the laser being poked at by multiple threads before you use ht is function call

4.109.3.8 void ArSick::configureShort (bool useSim = false, BaudRate baud = BAUD38400, Degrees deg = DEGREES180, Increment $incr = INCREMENT_ONE$)

Shorter configure for the laser.

You must lockDevice the laser or not have the laser being poked at by multiple threads before you use htis function call

4.109.3.9 bool ArSick::disconnect (bool doNotLockRobotForSim = false)

Disconnect from the laser.

Disconnects from the laser. You should lockDevice the laser before calling this function. Also if you are using the simulator it will lock the robot so it can send the command to the simulator, so you should make sure the robot is unlocked.

Parameters:

doNotLockRobotForSim if this is true, this will not lock the robot if its trying to send a command to the sim... ONLY do this if you are calling this from within the robots sync loop (ie from a sync task, sensor interp task, or user task)

Returns:

true if it could disconnect from the laser cleanly

4.109.3.10 void ArSick::filterReadings () [protected]

Internal function for filtering the raw readings and updating buffers.

filter readings here, from raw current buffer to filtered current buffer of the range device object, and then to the cumulative buffer

current buffer filtering is to eliminate max (null) readings, and compress close readings

cumulative buffer filtering is to replace readings within the scope of the current sensor set

4.109.3.11 int ArSick::getConnectionTimeoutTime (void)

Gets the time without a response until connection assumed lost.

Gets the number of seconds to go without response from the laser until it is assumed that the connection with the laser has been broken and the disconnect on error events will happen.

4.109.3.12 double ArSick::getFilterNearDist (void)

Current readings closer than this are discarded as too close.

When readings are put into the current buffer they are compared against the last reading and must be at least this distance away from the last reading. If this value is 0 then there is no filtering of this kind.

4.109.3.13 int ArSick::internalConnectHandler (void)

An internal function, single loop event to connect to laser.

Returns:

0 if its still trying to connect, 1 if it connected, 2 if it failed

4.109.3.14 bool ArSick::internalConnectSim (void)

An internal function.

Sends the commands to the sim to start up the connection

Returns:

true if the commands were sent, false otherwise

4.109.3.15 void ArSick::remConnectCB (ArFunctor * functor)

Adds a disconnect callback.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addConnectCB (p. 437)

4.109.3.16 void ArSick::remDisconnectNormallyCB (ArFunctor * functor)

Removes a callback for when disconnect is called while connected.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addDisconnectNormallyCB (p. 438)

4.109.3.17 void ArSick::remDisconnectOnErrorCB (ArFunctor * functor)

Removes a callback for when disconnection happens because of an error.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addDisconnectOnErrorCB (p. 438)

4.109.3.18 void ArSick::remFailedConnectCB (ArFunctor * functor)

Removes a callback for when a connection to the robot is failed.

Parameters:

functor the functor to remove from the list of connect callbacks

See also:

addFailedConnectCB (p. 439)

4.109.3.19 bool ArSick::runOnRobot (void)

Runs the laser off of the robot.

This sets up a sensor interp task on the robot, which is where the robot will be driven from. Note that the device must have been added to the robot already so that the device has a pointer to the robot. You should lock the robot and lockDevice the laser before doing this if other things are running already.

4.109.3.20 void ArSick::setConnectionTimeoutTime (int mSecs)

Sets the time without a response until connection assumed lost.

Sets the number of seconds to go without a response from the laser until it is assumed that the connection with the laser has been broken and the disconnect on error events will happen.

Parameters:

seconds if seconds is 0 then the connection timeout feature will be disabled, otherwise disconnect on error will be triggered after this number of seconds...

4.109.3.21 void ArSick::setFilterNearDist (double dist)

Current readings closer than this are discarded as too close.

When readings are put into the current buffer they are compared against the last reading and must be at least this distance away from the last reading. If this value is 0 then there is no filtering of this kind.

The documentation for this class was generated from the following files:

- ArSick.h
- ArSick.cpp

4.110 ArSickLogger Class Reference

This class can be used to create log files for the laser mapper.

#include <ArSickLogger.h>

Public Methods

• ArSickLogger (ArRobot *robot, ArSick *sick, double distDiff, double degDiff, const char *filename, bool addGoals=false, ArJoyHandler *joy-Handler=NULL)

Constructor.

• virtual ~ArSickLogger ()

Destructor.

• void addTagToLog (const char *str,...)

Adds a string to the log file at the given moment.

• void **addTagToLogPlain** (const char *str)

Same ass addToLog, but no varargs, wrapper for java.

• void **setDistDiff** (double distDiff)

Sets the distance at which the robot will take a new reading.

 \bullet double $\mathbf{getDistDiff}$ (void)

Gets the distance at which the robot will take a new reading.

• void **setDegDiff** (double degDiff)

Sets the degrees to turn at which the robot will take a new reading.

• double **getDegDiff** (void)

Gets the degrees to turn at which the robot will take a new reading.

• void takeReading (void)

Explicitly tells the robot to take a reading.

• void addGoal (void)

Adds a goal where the robot is at the moment.

• void **robotTask** (void)

The task which gets attached to the robot.

4.110.1 Detailed Description

This class can be used to create log files for the laser mapper.

This class has a pointer to a robot and a laser... every time the robot has EITHER moved the distDiff, or turned the degDiff, it will take the current readings from the laser and log them into the log file given as the filename to the constructor. Readings can also be taken by calling takeReading which explicitly tells the logger to take a reading.

The class can also add goals, see the constructor arg addGoals for information about that... you can also explicitly have it add a goal by calling addGoal.

4.110.2 Constructor & Destructor Documentation

4.110.2.1 ArSickLogger::ArSickLogger (ArRobot * robot, ArSick * sick, double distDiff, double degDiff, const char * filename, bool addGoals = false, ArJoyHandler * joyHandler = NULL)

Constructor.

Make sure you have called **ArSick::configure** (p. 440) or **ArSick::configure-Short** (p. 440) on your laser before you make this class

Parameters:

robot The robot to attach to
sick the laser to log from
distDiff the distance traveled at which to take a new reading
degDiff the degrees turned at which to take a new reading
filename the file name in which to put the log

addGoals whether to add goals automatically or... if true then the sick logger puts hooks into places it needs this to happen, into any key-handler thats around (for a keypress of G), it pays attention to the flag bit of the robot, and it puts in a button press callback for the joyhandler passed in (if any)

4.110.3 Member Function Documentation

4.110.3.1 void ArSickLogger::addTagToLog (const char * str, ...)

Adds a string to the log file at the given moment.

The robot MUST be locked before you call this function, so that this function is not adding to a list as the robotTask is using it.

This function takes the given tag

The documentation for this class was generated from the following files:

- \bullet ArSickLogger.h
- ArSickLogger.cpp

4.111 ArSickPacket Class Reference

Represents the packets sent to the sick as well as those received from it.

#include <ArSickPacket.h>

Inheritance diagram for ArSickPacket::



Public Methods

- ArSickPacket (unsigned char sendingAddress=0)

 Constructor.
- virtual \sim **ArSickPacket** ()

Destructor.

- void **setSendingAddress** (unsigned char address)

 Sets the address to send this packet to (only use for sending).
- unsigned char **getSendingAddress** (void)

 Sets the address to send this packet to (only use for sending).
- unsigned char **getReceivedAddress** (void)

 Gets the address this packet was sent from (only use for receiving).
- bool verifyCRC (void)

returns true if the crc matches what it should be.

• ArTypes::UByte getID (void)

returns the ID of the packet (first byte of data).

• ArTypes::Byte2 calcCRC (void)

returns the crc, probably used only internally.

• virtual void finalizePacket (void)

MakeFinals the packet in preparation for sending, must be done.

• virtual void **resetRead** (void)

Restart the reading process.

• ArTime getTimeReceived (void)

Gets the time the packet was received at.

• void **setTimeReceived** (**ArTime** timeReceived)

Sets the time the packet was received at.

• virtual void **duplicatePacket** (ArSickPacket *packet)

Duplicates the packet.

4.111.1 Detailed Description

Represents the packets sent to the sick as well as those received from it.

This class reimplements some of the buf operations since the robot is little endian.

You can just look at the documentation for the **ArBasePacket** (p. 112) except for these functions here, setAddress, getAddress, verifyCheckSum, print, getID, and calcCheckSum.

4.111.2 Member Function Documentation

4.111.2.1 void ArSickPacket::duplicatePacket (ArSickPacket * packet) [virtual]

Duplicates the packet.

Copies the given packets buffer into the buffer of this packet, also sets this length and readlength to what the given packet has

Parameters:

packet the packet to duplicate

4.111.2.2 unsigned char ArSickPacket::getReceivedAddress (void)

Gets the address this packet was sent from (only use for receiving).

This gets the address that this packet was received from. Note that this is only valid if this packet was received from a laser, if you want to know where a packet was addressed to use getSendingAdress instead.

Returns:

the address a packet was received from

4.111.2.3 unsigned char ArSickPacket::getSendingAddress (void)

Sets the address to send this packet to (only use for sending).

This gets the address for use in sending packets, the address is what has been saved, then when a packet is finalizePacketd for sending, the address is put into the appropriate spot in the packet.

Returns:

the address of the laser to be addressed

4.111.2.4 void ArSickPacket::resetRead (void) [virtual]

Restart the reading process.

Sets the length read back to the header length so the packet can be reread using the other methods

Reimplemented from ArBasePacket (p. 117).

4.111.2.5 void ArSickPacket::setSendingAddress (unsigned char address)

Sets the address to send this packet to (only use for sending).

This sets the address for use in sending packets, the address is saved, then when a packet is finalizePacketd for sending, the address is put into the appropriate spot in the packet.

Parameters:

address the address of the laser to be addressed

The documentation for this class was generated from the following files:

- ArSickPacket.h
- ArSickPacket.cpp

4.112 ArSickPacketReceiver Class Reference

Given a device connection it receives packets from the sick through it.

#include <ArSickPacketReceiver.h>

Public Methods

• ArSickPacketReceiver (unsigned char receivingAddress=0, bool allocatePackets=false, bool useBase0Address=false)

Constructor without an already assigned device connection.

• ArSickPacketReceiver (ArDeviceConnection *deviceConnection, unsigned char receivingAddress=0, bool allocatePackets=false, bool use-Base0Address=false)

Constructor with assignment of a device connection.

• virtual ~ArSickPacketReceiver ()

Destructor.

• ArSickPacket * receivePacket (unsigned int msWait=0)

Receives a packet from the robot if there is one available.

• void **setDeviceConnection** (**ArDeviceConnection** *device-Connection)

Sets the device this instance receives packets from.

• ArDeviceConnection * getDeviceConnection (void)

Gets the device this instance receives packets from.

• bool isAllocatingPackets (void)

Gets whether or not the receiver is allocating packets.

4.112.1 Detailed Description

Given a device connection it receives packets from the sick through it.

4.112.2 Constructor & Destructor Documentation

4.112.2.1 ArSickPacketReceiver:::ArSickPacketReceiver (unsigned char receivingAddress = 0, bool allocatePackets = false, bool useBase0Address = false)

Constructor without an already assigned device connection.

Parameters:

allocatePackets whether to allocate memory for the packets before returning them (true) or to just return a pointer to an internal packet (false)... most everything should use false as this will help prevent many memory leaks or corruptions

4.112.2.2 ArSickPacketReceiver::ArSickPacketReceiver (ArDeviceConnection * deviceConnection, unsigned char receivingAddress = 0, bool allocatePackets = false, bool useBase0Address = false)

Constructor with assignment of a device connection.

Parameters:

deviceConnection the connection which the receiver will use

allocatePackets whether to allocate memory for the packets before returning them (true) or to just return a pointer to an internal packet (false)... most everything should use false as this will help prevent many memory leaks or corruptions

4.112.3 Member Function Documentation

4.112.3.1 ArSickPacket * ArSickPacketReceiver::receivePacket (unsigned int msWait = 0)

Receives a packet from the robot if there is one available.

Parameters:

ms Wait how long to block for the start of a packet, nonblocking if 0

Returns:

NULL if there are no packets in alloted time, otherwise a pointer to the packet received, if allocatePackets is true than the place that called this function owns the packet and should delete the packet when done... if allocatePackets is false then nothing must store a pointer to this packet,

the packet must be used and done with by the time this method is called again

The documentation for this class was generated from the following files:

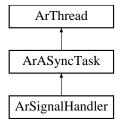
- \bullet ArSickPacketReceiver.h
- $\bullet \ Ar Sick Packet Receiver.cpp$

4.113 ArSignalHandler Class Reference

Signal handling class.

#include <ArSignalHandler.h>

Inheritance diagram for ArSignalHandler::



Public Methods

• virtual ~ArSignalHandler ()

Destructor.

• virtual void * runThread (void *arg)

The main run loop.

Static Public Methods

• void createHandlerNonThreaded ()

Setup the signal handling for a non-threaded program.

• void createHandlerThreaded ()

 $Setup\ the\ signal\ handling\ for\ a\ multi-threaded\ program.$

• void **blockCommon** ()

Block all the common signals the kill a program.

• void unblockAll ()

Unblock all the signals.

• void **block** (Signal sig)

Block the given signal.

• void **unblock** (Signal sig)

Unblock the given signal.

• void handle (Signal sig)

Handle the given signal.

• void unhandle (Signal sig)

Dont handle the given signal.

void addHandlerCB (ArFunctor1< int > *func, ArListPos::Pos position)

Add a handler callback.

• void **delHandlerCB** (**ArFunctor1**< int > *func)

Remove a handler callback.

• ArSignalHandler * getHandler ()

Get a pointer to the single ArSignalHandler instance.

• std::string nameSignal (int sig)

Get the name of the given signal.

• void blockCommonThisThread ()

Block all the common signals for the calling thread only.

• void blockAllThisThread ()

Block all the signals for the calling thread only.

4.113.1 Detailed Description

Signal handling class.

This is a signal handling class. It has both a threaded and non-threaded mode of operation. The non-threaded mode will work in a threaded application but it is best to use the threaded mode. The benefit of the threaded mode is that if the signal incures some processing, but does not shutdown the program (ie. SIGUSR1 or SIGUSR2), the threaded mode will handle the signal in its own thread and hopefully that will not hurt the performance of the tight loop robot control. Exactly how much performance you get out of this depends on your machines physical hardware and exactly what the processing the signal handler does. For instance, a multi-processor machine has a much greater chance of the signal handler not interfering with the robot control loop.

See the **Aria** (p. 205) main class for how to initialize a default setup of the signal handling.

There are functions to block, unblock, handle and unhandle signals. These functions all must be called before creating the signalhandler. In either single or multi-threaded mode. The functions to block and handle signals creates a set of blocking and handling which is then used by the create functions to tell the Linux kernel what to do.

In the threaded mode, there is a signal handler thread that is created. That thread is created in a detached state, which means it can not be joined on. When the program exits, the signal handler thread will be ignored and that thread will never exit its run loop. This is perfectly fine behavior. There is no state that can be messed up in this fashion. It is just easier to exit the program than to try to wake up that thread and get it to exit itself.

This class is for Linux only. Windows has virtually no support for signals and the little support that it does have is not realy usefull. There is an empty implementation of this class for Windows so that code can compile in both Linux and Windows. Just do not expect the code that uses this signal handling to do anything in Windows. This should not be a problem since signals are not used in Windows.

4.113.2 Member Function Documentation

4.113.2.1 void ArSignalHandler::addHandlerCB (ArFunctor1< int > * func, ArListPos::Pos position) [static]

Add a handler callback.

Add a handler callback to the list of callbacks. When there is a signal sent to the process, the list of callbacks are invoked and passed the signal number.

Parameters:

functor functor created from ArFunctorC1<int> which refers to the function to call.

position whether to place the functor first or last

4.113.2.2 void ArSignalHandler::block (Signal sig) [static]

Block the given signal.

Block the given signal. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

Parameters:

siq the number of the signal

4.113.2.3 void ArSignalHandler::blockCommon () [static]

Block all the common signals the kill a program.

Sets the signal handler to block all the common signals. The 'common' signals are SIGHUP, SIGINT, SIGQUIT, SIGTERM, SIGSEGV, and SIGPIPE. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

4.113.2.4 void ArSignalHandler::blockCommonThisThread () [static]

Block all the common signals for the calling thread only.

Block all the common signals for the calling thread. The calling thread will never recieve the common signals which are SIGHUP, SIGINT, SIGQUIT, and SIGTERM. This function can be called at any time.

4.113.2.5 void ArSignalHandler::createHandlerNonThreaded () [static]

Setup the signal handling for a non-threaded program.

Sets up the signal handling for a non-threaded program. When the program This uses the system call signal(2). This should not be used if you have a threaded program.

See also:

createHandlerThreaded (p. 456)

4.113.2.6 void ArSignalHandler::createHandlerThreaded () [static]

Setup the signal handling for a multi-threaded program.

Sets up the signal handling for a non-threaded program. This call is only usefull for Linux. This will create a dedicated thread in which to handle signals. The thread calls sigwait(3) and waits for a signal to be sent. By default all **Ar-Thread** (p. 487) instances block all signals. Thus the signal is sent to the signal handler thread. This will allow the other threads to continue uninterrupted and not skew their timing loops.

See also:

createHandlerNonThreaded (p. 456)

4.113.2.7 void ArSignalHandler::delHandlerCB (ArFunctor1< int > * func) [static]

Remove a handler callback.

Remove a handler callback from the list of callbacks.

Parameters:

functor functor created from ArFunctorC1<int> which refers to the function to call.

4.113.2.8 ArSignalHandler * ArSignalHandler::getHandler () [static]

Get a pointer to the single ArSignalHandler instance.

Get a pointer to the single instance of the ArSignalHandler. The signal handler uses the singleton model, which means there can only be one instance of ArSignalHandler. If the single instance of ArSignalHandler has not been created, getHandler will create it. This is how the handler should be created.

Returns:

returns a pointer to the instance of the signal handler

4.113.2.9 void ArSignalHandler::handle (Signal sig) [static]

Handle the given signal.

Handle the given signal. All the handler callbacks will be called with this signal when it is recieved. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

Parameters:

sig the number of the signal

$\begin{array}{ll} \textbf{4.113.2.10} & \textbf{void} * \textbf{ArSignalHandler::runThread} \; (\textbf{void} * \textit{arg}) \\ & [\texttt{virtual}] \end{array}$

The main run loop.

Override this function and put your taskes run loop here. Check the value of **getRunning**() (p. 488) or myRunning periodicly in your loop. If the value goes false, the loop should exit and **runThread**() (p. 457) should return.

Reimplemented from ArASyncTask (p. 111).

4.113.2.11 void ArSignalHandler::unblock (Signal sig) [static]

Unblock the given signal.

Unblock the given signal. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

Parameters:

sig the number of the signal

4.113.2.12 void ArSignalHandler::unblockAll () [static]

Unblock all the signals.

Unblock all the signals. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

4.113.2.13 void ArSignalHandler::unhandle (Signal sig) [static]

Dont handle the given signal.

Do not handle the given signal. Call this before calling create HandlerNon-Threaded or create HandlerThreaded.

Parameters:

sig the number of the signal

The documentation for this class was generated from the following files:

- ArSignalHandler.h
- ArSignalHandler_LIN.cpp
- ArSignalHandler_WIN.cpp

4.114 ArSimpleConnector Class Reference

This class simplifies connecting to the robot and/or laser.

#include <ArSimpleConnector.h>

Public Methods

• ArSimpleConnector (int *argc, char **argv)

Constructor that takes args from the main.

• ArSimpleConnector (ArArgumentBuilder *builder)

Constructor that takes argument builder.

• ~ArSimpleConnector (void)

Destructor.

• bool **setupRobot** (**ArRobot** *robot)

Sets up the robot to be connected.

• bool connectRobot (ArRobot *robot)

Sets up the robot then connects it.

• bool setupLaser (ArSick *sick)

Sets up the laser to be connected.

• void parseArgs (void)

Function to parse the arguments given.

• void logOptions (void) const

Log the options the simple connector has.

4.114.1 Detailed Description

This class simplifies connecting to the robot and/or laser.

First of all, when you create your ArSimpleConnector you pass it either the argc and argv from main or you can pass it an **ArArgumentBuilder** (p. 106), which you might do if you were making a windows executable that had a WinMain instead of a main.

Then you need to tell it to parseArgs, which parses those arguments in order to know where to connect the robot and/or laser.

Then you can either set up the robot to be connected with setupRobot or just connect it with connectRobot. You'll still need to run or runAsync the robot.

You can then set up the laser with setupLaser, but you'll have to run or run-Async the laser and connect it yourself.

4.114.2 Member Function Documentation

4.114.2.1 bool ArSimpleConnector::setupLaser (ArSick * sick)

Sets up the laser to be connected.

Description of the logic for connection to the laser. If –remoteHost then the laser will a tcp connection will be opened to that remoteHost at port 8102 or –remoteLaserTcpPort if that argument is given, if this connection fails then the setup fails. If –remoteHost wasn't provided and the robot connected to a simulator as described elsewhere then the laser is just configured to be simulated, if the robot isn't connected to a simulator it tries to open a serial connection to ArUtil::COM3 or –laserPort if that argument is given.

4.114.2.2 bool ArSimpleConnector::setupRobot (ArRobot * robot)

Sets up the robot to be connected.

Description of the logic for connection to the robot. If –remoteHost is given then the connector tries to open a tcp connection to port 8101 by default, or –remote-RobotTcpPort if that was an option provided and if this tcp connection fails then the whole connection fails. If no remoteHost was given it first tries to open a tcp connection to localhost on port 8101 (or –remoteRoboTcpPort) which is the where the simulator runs, if this tcp connection succeeds then the connector assumes its connecting to the simulator, if this connection fails then it assumes a serial connection to the real robot is desired and connects to ArUtil::COM1 or –robotPort if that argument was supplied.

The documentation for this class was generated from the following files:

- ArSimpleConnector.h
- ArSimpleConnector.cpp

4.115 ArSocket Class Reference

socket communication wrapper.

#include <ArSocket.h>

Public Methods

• ArSocket ()

Constructor.

• **ArSocket** (const char *host, int port, Type type)

Constructor which connects to a server.

 \bullet $\mathbf{ArSocket}$ (int port, bool doClose, Type type)

Constructor which opens a server port.

• \sim ArSocket ()

Destructor.

• bool copy (int fd, bool doclose)

Copy socket structures.

• void copy (ArSocket *s)

 $Copy\ socket\ structures.$

• void transfer (ArSocket *s)

Transfer ownership of a socket.

• bool **connect** (const char *host, int port, Type type)

Connect as a client to a server.

• bool open (int port, Type type)

Open a server port.

• bool **create** (Type type)

Simply create a port.

• bool findValidPort (int startPort)

Find a valid unused port and bind the socket to it.

• bool connectTo (const char *host, int port)

Connect the socket to the given address.

- bool **connectTo** (struct sockaddr_in *sin)

 Connect the socket to the given address.
- bool accept (ArSocket *sock)

 Accept a new connection.
- bool close ()

 Close the socket.
- int write (const void *buff, size_t len)

 Write to the socket.
- int read (void *buff, size_t len, unsigned int msWait=0)

 Read from the socket.
- int sendTo (const void *msg, int len)

 Send a message on the socket.
- int **sendTo** (const void *msg, int len, struct sockaddr_in *sin)

 Send a message on the socket.
- int recvFrom (void *msg, int len, sockaddr_in *sin)

 Receive a message from the socket.
- bool **getSockName** ()

 Get the socket name. Stored in ArSocket::mySin.
- sockaddr_in * sockAddrIn ()

 Accessor for the sockaddr.
- in_addr * inAddr ()

 Accessor for the in_addr.
- unsigned short int **inPort** ()

 Accessor for the port of the sockaddr.
- bool **setLinger** (int time)

 Set the linger value.
- bool **setBroadcast** ()

 Set broadcast value.

• bool setReuseAddress ()

Set the reuse address value.

• bool setNonBlock ()

Set socket to nonblocking.

• void **setDoClose** (bool yesno)

 $Change\ the\ doClose\ value.$

• int **getFD** () const

Get the file descriptor.

• Type **getType** () const

Get the protocol type.

 \bullet const std::string & $\mathbf{getErrorStr}$ () const

Get the last error string.

• Error **getError** () const

Get the last error.

• int writeString (const char *str,...)

Writes a string to the socket (adding end of line characters).

• int writeStringPlain (const char *str)

Same as writeString, but no varargs, wrapper for java.

• bool readString (char *buf, size_t len)

Reads a string from the socket.

Static Public Methods

• bool init ()

Initialize the network layer.

• void shutdown ()

Shutdown the network layer.

• bool **hostAddr** (const char *host, struct in_addr &addr)

Convert a host string to an address structure.

- bool addrHost (struct in_addr &addr, char *host)

 Convert an address structure to a host string.
- std::string **getHostName** ()

 Get the localhost address.
- void **inToA** (struct in_addr *addr, char *buff)

 Convert addr into string numerical address.
- const size_t sockAddrLen ()

 Size of the sockaddr.
- const size_t maxHostNameLen ()

 Max host name length.
- unsigned int **hostToNetOrder** (int i)

 Host byte order to network byte order.
- unsigned int **netToHostOrder** (int i)

 Network byte order to host byte order.

Static Public Attributes

• bool **ourInitialized** = true

We're always initialized in Linux.

4.115.1 Detailed Description

socket communication wrapper.

ArSocket is a layer which allows people to use the sockets networking interface in an operating system independent manner. All of the standard commonly used socket functions are implemented. This class also contains the file descriptor which identifies the socket to the operating system.

In Windows, the networking subsystem needs to be initialized and shutdown individually by each program. So when a program starts they will need to call the static function **ArSocket::init()** (p. 465) and call **ArSocket::shutdown()** (p. 466) when it exits. For programs that use **Aria::init()** (p. 208) and **Aria::uninit()** (p. 209) calling the **ArSocket::init()** (p. 465) and **ArSocket::shutdown()** (p. 466) is unnecessary. The **Aria** (p. 205) initialization functions take care of this. These functions do nothing in Linux.

4.115.2 Constructor & Destructor Documentation

4.115.2.1 ArSocket::ArSocket (const char * host, int port, Type type)

Constructor which connects to a server.

Constructs the socket and connects it to the given host.

Parameters:

host hostname of the server to connect to
port port number of the server to connect to
type protocol type to use

4.115.2.2 ArSocket::ArSocket (int port, bool doClose, Type type)

Constructor which opens a server port.

Constructs the socket and opens it as a server port.

Parameters:

port port number to bind the socket to
doClose automaticaly close the port if the socket is destructed
type protocol type to use

4.115.3 Member Function Documentation

4.115.3.1 bool ArSocket::copy (int fd, bool doclose)

Copy socket structures.

Copy socket structures. Copy from one Socket to another will still have the first socket close the file descripter when it is destructed.

4.115.3.2 bool ArSocket::init (void) [static]

Initialize the network layer.

In Windows, the networking subsystem needs to be initialized and shutdown individually by each program. So when a program starts they will need to call the static function **ArSocket::init()** (p. 465) and call **ArSocket::shutdown()** (p. 466) when it exits. For programs that use **Aria::init()** (p. 208) and **Aria::uninit()** (p. 209) calling the **ArSocket::init()** (p. 465) and **ArSocket::shutdown()** (p. 466) is unnecessary. The **Aria** (p. 205) initialization functions take care of this. These functions do nothing in Linux.

4.115.3.3 int ArSocket::read (void * buff, size_t len, unsigned int msWait = 0) [inline]

Read from the socket.

Parameters:

buff buffer to read into

len how many bytes to read

msWait if 0, don't block, if > 0 wait this long for data

Returns:

number of bytes read

4.115.3.4 bool ArSocket::readString (char * buf, size_t len) [inline]

Reads a string from the socket.

Parameters:

buf the buffer to read the string into, if there is no error but there is no string to read then the first character of the buffer is set to the null character

len the lenth of the buffer

Returns:

true if the socket could be read from, false if it couldn't (which also means the connection should be closed)

4.115.3.5 void ArSocket::shutdown () [static]

Shutdown the network layer.

In Windows, the networking subsystem needs to be initialized and shutdown individually by each program. So when a program starts they will need to call the static function **ArSocket::init()** (p. 465) and call **ArSocket::shutdown()** (p. 466) when it exits. For programs that use **Aria::init()** (p. 208) and **Aria::uninit()** (p. 209) calling the **ArSocket::init()** (p. 465) and **ArSocket::shutdown()** (p. 466) is unnecessary. The **Aria** (p. 205) initialization functions take care of this. These functions do nothing in Linux.

4.115.3.6 void ArSocket::transfer (ArSocket * s) [inline]

Transfer ownership of a socket.

transfer() (p. 467) will transfer ownership to this socket. The input socket will no longer close the file descriptor when it is destructed.

4.115.3.7 int ArSocket::write (const void * buff, size_t len) [inline]

Write to the socket.

Parameters:

buff buffer to write fromlen how many bytes to write

Returns:

number of bytes written

4.115.3.8 int ArSocket::writeString (const char * str, ...) [inline]

Writes a string to the socket (adding end of line characters).

This cannot write more than 2048 number of bytes

Parameters:

str the string to write to the socket

Returns:

number of bytes written

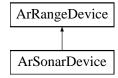
- ArSocket.h
- ArSocket_LIN.cpp
- ArSocket_WIN.cpp

4.116 ArSonarDevice Class Reference

A class for keeping track of sonar.

#include <ArSonarDevice.h>

Inheritance diagram for ArSonarDevice::



Public Methods

• **ArSonarDevice** (size_t currentBufferSize=24, size_t cumulativeBufferSize=64, const char *name="sonar")

Constructor.

• ~ArSonarDevice ()

Destructor.

• void **processReadings** (void)

Grabs the new readigns from the robot and adds them to the buffers.

• virtual void **setRobot** (**ArRobot** *robot)

 $Sets\ the\ robot\ pointer,\ also\ attaches\ its\ process\ function\ to\ the\ sensorInterp$ of the robot.

• virtual void addReading (double x, double y)

Adds sonar readings to the current and cumulative buffers Overrides the **ArRangeDevice** (p. 312) default action.

• void **setCumulativeMaxRange** (double r)

Maximum range for a reading to be added to the cumulative buffer (mm).

4.116.1 Detailed Description

A class for keeping track of sonar.

This class is for keeping a sonar history, and using that for obstacle avoidance and displays and what not

4.116.2 Member Function Documentation

4.116.2.1 void ArSonarDevice::addReading (double x, double y) [virtual]

Adds sonar readings to the current and cumulative buffers Overrides the **Ar-RangeDevice** (p. 312) default action.

Adds a sonar reading with the global coordinates x,y. Makes sure the reading is within the proper distance to the robot, for both current and cumulative buffers. Filters buffer points Note: please lock the device using **lockDevice()** (p. 318) / **unlockDevice()** (p. 319) if calling this from outside process().

Parameters:

- \boldsymbol{x} the global x coordinate of the reading
- y the global y coordinate of the reading

Reimplemented from ArRangeDevice (p. 312).

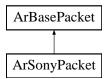
- ArSonarDevice.h
- ArSonarDevice.cpp

4.117 ArSonyPacket Class Reference

A class for for making commands to send to the sony.

#include <ArSonyPTZ.h>

Inheritance diagram for ArSonyPacket::



Public Methods

- ArSonyPacket (ArTypes::UByte2 bufferSize=15)

 Constructor.
- virtual void **uByteToBuf** (**ArTypes::UByte** val)

 Puts **ArTypes::UByte** (p. 496) into packets buffer.
- virtual void byte2ToBuf (ArTypes::Byte2 val)
 Puts ArTypes::Byte2 (p. 496) into packets buffer.
- void byte2ToBufAtPos (ArTypes::Byte2 val, ArTypes::UByte2 pose)

This is a new function, read the details before you try to use it.

4.117.1 Detailed Description

A class for for making commands to send to the sony.

There are only two functioning ways to put things into this packet, you MUST use thse, if you use anything else your commands won't work. You must use uByteToBuf and byte2ToBuf.

4.117.2 Member Function Documentation

4.117.2.1 void ArSonyPacket::byte2ToBufAtPos (ArTypes::Byte2 val, ArTypes::UByte2 pose)

This is a new function, read the details before you try to use it.

This function is my concession to not rebuilding a packet from scratch for every command, basicaly this is to not lose all speed over just using a character array. This is used by the default sony commands, unless you have a deep understanding of how the packets are working and what the packet structure looks like you should not play with this function, it also isn't worth it unless you'll be sending commands frequently.

Parameters:

val the Byte2 to put into the packet
pose the position in the packets array to put the value

- ArSonyPTZ.h
- ArSonyPTZ.cpp

4.118 ArSonyPTZ Class Reference

A class to use the sony pan tilt zoom unit.

#include <ArSonyPTZ.h>

Inheritance diagram for ArSonyPTZ::



Public Types

• enum { $MAX_PAN = 95$, $MAX_TILT = 25$, $MIN_ZOOM = 0$, $MAX_ZOOM = 1023$ }

Public Methods

- virtual bool **init** (void)

 Initializes the camera.
- virtual bool **pan** (int degrees)

 Pans to the given degrees.
- virtual bool panRel (int degrees)
 Pans relative to current position by given degrees.
- virtual bool **tilt** (int degrees)

 Tilts to the given degrees.
- virtual bool **tiltRel** (int degrees)

 Tilts relative to the current position by given degrees.
- virtual bool **panTilt** (int degreesPan, int degreesTilt)

 Pans and tilts to the given degrees.
- virtual bool **panTiltRel** (int degreesPan, int degreesTilt)

 Pans and tilts relatives to the current position by the given degrees.
- virtual bool canZoom (void) const

Returns true if camera can zoom (or rather, if it is controlled by this).

• virtual bool **zoom** (int zoomValue)

Zooms to the given value.

• virtual bool **zoomRel** (int zoomValue)

Zooms relative to the current value, by the given value.

• virtual int getPan (void) const

The angle the camera was last told to pan to.

• virtual int **getTilt** (void) const

The angle the camera was last told to tilt to.

• virtual int getZoom (void) const

The value the camera was last told to zoom to.

• virtual int getMaxPosPan (void) const

Gets the highest positive degree the camera can pan to.

• virtual int **getMaxNegPan** (void) const

Gets the lowest negative degree the camera can pan to.

• virtual int **getMaxPosTilt** (void) const

Gets the highest positive degree the camera can tilt to.

• virtual int getMaxNegTilt (void) const

Gets the lowest negative degree the camera can tilt to.

• virtual int **getMaxZoom** (void) const

Gets the maximum value for the zoom on this camera.

• virtual int **getMinZoom** (void) const

Gets the lowest value for the zoom on this camera.

4.118.1 Detailed Description

A class to use the sony pan tilt zoom unit.

4.118.2 Member Enumeration Documentation

4.118.2.1 anonymous enum

Enumeration values:

MAX_PAN maximum degrees the unit can pan (either direction).

MAX_TILT maximum degrees the unit can tilt (either direction).

MIN_ZOOM minimum value for zoom.

MAX_ZOOM maximum value for zoom.

- ArSonyPTZ.h
- ArSonyPTZ.cpp

4.119 ArSyncTask Class Reference

Class used internally to manage the functions that are called every cycle.

#include <ArSyncTask.h>

Public Methods

• ArSyncTask (const char *name, ArFunctor *functor=NULL, ArTask-State::State *state=NULL, ArSyncTask *parent=NULL)

Constructor, shouldn't ever do a new on anything besides the root node.

• virtual ~**ArSyncTask** ()

Destructor.

• void **run** (void)

Runs the node, which runs all children of this node as well.

• void log (int depth=0)

Prints the node, which prints all the children of this node as well.

• ArTaskState::State getState (void)

Gets the state of the task.

• void setState (ArTaskState::State state)

Sets the state of the task.

• ArSyncTask * findNonRecursive (const char *name)

Finds the task in the instances list of children, by name.

• ArSyncTask * findNonRecursive (ArFunctor *functor)

Finds the task in the instances list of children, by functor.

• ArSyncTask * find (const char *name)

Finds the task recursively down the tree by name.

• ArSyncTask * find (ArFunctor *functor)

Finds the task recursively down the tree by functor.

void addNewBranch (const char *nameOfNew, int position, ArTask-State::State *state=NULL)

Adds a new branch to this instance.

• void addNewLeaf (const char *nameOfNew, int position, ArFunctor *functor, ArTaskState::State *state=NULL)

Adds a new leaf to this instance.

- std::string **getName** (void)

 Gets the name of this task.
- ArFunctor * getFunctor (void)

Gets the functor this instance runs, if there is one.

4.119.1 Detailed Description

Class used internally to manage the functions that are called every cycle.

This is used internally, no user should ever have to create one, but serious developers may want to use the members. Most users will be able to use the user tasks defined in the **ArRobot** (p. 355) class. This class should only be used by serious developers.

The way it works is that each instance is a node in a tree. The only node that should ever be created with a new is the top one. The run and print functions both call the run/print on themselves, then on all of their children, going from lowest numbered position to highest numbered, lower going first. There are no hard limits to the position, it can be any integer. ARIA uses the convention of 0 to 100, when you add things of your own you should leave room to add in between. Also you can add things with the same position, the only effect this has is that the first addition will show up first in the run or print.

After the top one is created, every other task should be created with either add-NewBranch or addNewLeaf. Each node can either be a branch node or a list node. The list (multimap actually) of branches/nodes is ordered by the position passed in to the add function. addNewBranch adds a new branch node to the instance it is called on, with the given name and position. addNewLeaf adds a new leaf node to the instance it is called on, with the given name and position, and also with the **ArFunctor** (p. 139) given, this functor will be called when the leaf is run. Either add creates the new instance and puts it in the list of branches/nodes in the approriate spot.

The tree takes care of all of its own memory management and list management, the add functions put into the list and creates the memory, conversely if you delete an ArSyncTask (which is the correct way to get rid of one) it will remove itself from its parents list.

If you want to add something to the tree the proper way to do it is to get the pointer to the root of the tree (ie with ArRobot::getSyncProcRoot) and then to use find on the root to find the branch you want to travel down, then continue this until you find the node you want to add to. Once there just call addNewBranch or addNewLeaf and you're done.

There is now a pointer to an integer that is the state of the task, if this pointer is given whenever something changes the state of the task it will modify the value pointed to. If the pointer is NULL then the syncTask will use an integer of its own to keep track of the state of the process.

4.119.2 Constructor & Destructor Documentation

4.119.2.1 ArSyncTask::ArSyncTask (const char * name, ArFunctor * functor = NULL, ArTaskState::State * state = NULL, ArSyncTask * parent = NULL)

Constructor, shouldn't ever do a new on anything besides the root node.

New should never be called to create an ArSyncTask except to create the root node. Read the detailed documentation of the class for details.

4.119.2.2 ArSyncTask::~ArSyncTask () [virtual]

Destructor.

If you delete the task it deletes everything in its list, so to delete the whole tree just delete the top one... also note that if you delete a node, it will remove itself from its parents list.

4.119.3 Member Function Documentation

4.119.3.1 void ArSyncTask::addNewBranch (const char * nameOfNew, int position, ArTaskState::State * state = NULL)

Adds a new branch to this instance.

Creates a new task with the given name and puts the task into its own iternal list at the given position.

Parameters:

nameOfNew Name to give to the new task.

position place in list to put the branch, things are run/printed in the order of highest number to lowest number, no limit on numbers (other than that it is an int). ARIA uses 0 to 100 just as a convention.

4.119.3.2 void ArSyncTask::addNewLeaf (const char * nameOfNew, int position, ArFunctor * functor, ArTaskState::State * state = NULL)

Adds a new leaf to this instance.

Creates a new task with the given name and puts the task into its own iternal list at the given position. Sets the nodes functor so that it will call the functor when run is called.

Parameters:

nameOfNew Name to give to the new task.

position place in list to put the branch, things are run/printed in the order of highest number to lowest number, no limit on numbers (other than that it is an int). ARIA uses 0 to 100 just as a convention.

functor ArFunctor (p. 139) which contains the functor to invoke when run is called.

4.119.3.3 ArSyncTask * ArSyncTask::find (ArFunctor * functor)

Finds the task recursively down the tree by functor.

Finds a node below (or at) this level in the tree with the given name

Parameters:

name The name of the child we are interested in finding

Returns:

The task, if found. If not found, NULL.

4.119.3.4 ArSyncTask * ArSyncTask::find (const char * name)

Finds the task recursively down the tree by name.

Finds a node below (or at) this level in the tree with the given name

Parameters:

name The name of the child we are interested in finding

Returns:

The task, if found. If not found, NULL.

4.119.3.5 ArSyncTask * ArSyncTask::findNonRecursive (ArFunctor * functor)

Finds the task in the instances list of children, by functor.

Finds a child of this node with the given functor

Parameters:

functor the functor we are interested in finding

Returns:

The task, if found. If not found, NULL.

4.119.3.6 ArSyncTask * ArSyncTask::findNonRecursive (const char * name)

Finds the task in the instances list of children, by name.

Finds a child of this node with the given name

Parameters:

name The name of the child we are interested in finding

Returns:

The task, if found. If not found, NULL.

4.119.3.7 void ArSyncTask::log (int depth = 0)

Prints the node, which prints all the children of this node as well.

Prints the node... the defaulted depth parameter controls how far over to print the data (how many tabs)... it recurses down all its children.

4.119.3.8 void ArSyncTask::run (void)

Runs the node, which runs all children of this node as well.

If this node is a leaf it calls the functor for the node, if it is a branch it goes through all of the children in the order of highest position to lowest position and calls run on them.

- ArSyncTask.h
- ArSyncTask.cpp

4.120 ArTaskState Class Reference

Class with the different states a task can be in.

#include <ArTaskState.h>

Public Types

• enum State { INIT = 0, RESUME, ACTIVE, SUSPEND, SUCCESS, FAILURE, USER_START = 20 }

4.120.1 Detailed Description

Class with the different states a task can be in.

These are the defined states, if the state is anything other than is defined here that is annotated (not running) the process will be run. No one should have any of their own states less than the USER_START state. People's own states should start at USER_START or at USER_START plus a constant (so they can have different sets of states).

4.120.2 Member Enumeration Documentation

4.120.2.1 enum ArTaskState::State

Enumeration values:

INIT Initialized (running).

RESUME Resumed after being suspended (running).

ACTIVE Active (running).

SUSPEND Suspended (not running).

SUCCESS Succeeded and done (not running).

FAILURE Failed and done (not running).

USER_START This is where the user states should start (they will all be run).

The documentation for this class was generated from the following file:

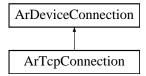
• ArTaskState.h

4.121 ArTcpConnection Class Reference

For connectiong to a device through a socket.

#include <ArTcpConnection.h>

Inheritance diagram for ArTcpConnection::



Public Types

• enum Open { OPEN_NET_FAIL = 1, OPEN_BAD_HOST, OPEN_NO_ROUTE, OPEN_CON_REFUSED }

Public Methods

• ArTcpConnection ()

Constructor.

• virtual ~ArTcpConnection ()

Destructor also closes connection.

- int open (const char *host=NULL, int port=8101)

 Opens a connection to the given host and port.
- virtual bool **openSimple** (void)

Opens the connection again, using the values from setLocation or.

• virtual int **getStatus** (void)

Gets the status of the connection, which is one of the enum status.

• virtual bool close (void)

Closes the connection.

• virtual int **read** (const char *data, unsigned int size, unsigned int ms-Wait=0)

Reads data from connection.

- virtual int write (const char *data, unsigned int size)

 Writes data to connection.
- virtual const char * **getOpenMessage** (int messageNumber)

 Gets the string of the message associated with opening the device.
- virtual ArTime getTimeRead (int index)

Gets the time data was read in.

- virtual bool isTimeStamping (void)
 sees if timestamping is really going on or not.
- std::string **getHost** (void)

 Gets the name of the host connected to.
- int getPort (void)

 Gets the number of the port connected to.
- int internalOpen (void)

 Internal function used by open and openSimple.
- void setSocket (ArSocket *socket)

 Sets the tcp connection to use this socket instead of its own.
- ArSocket * getSocket (void)
 Gets the socket this tcp connection is using.
- ullet void $\mathbf{setStatus}$ (int status)

Sets the status of the device, ONLY use this if you're playing with setSocket and know what you're doing.

4.121.1 Detailed Description

For connectiong to a device through a socket.

4.121.2 Member Enumeration Documentation

4.121.2.1 enum ArTcpConnection::Open

Enumeration values:

OPEN_NET_FAIL Some critical part of the network isn't working.

OPEN_BAD_HOST Could not find the host.

OPEN_NO_ROUTE Know where the host is, but can't get to it.

OPEN_CON_REFUSED Got to the host but it didn't allow a connec-

4.121.3 Member Function Documentation

4.121.3.1 bool ArTcpConnection::close (void) [virtual]

Closes the connection.

tion.

Returns:

whether the close succeeded or not

Reimplemented from ArDeviceConnection (p. 126).

4.121.3.2 std::string ArTcpConnection::getHost (void)

Gets the name of the host connected to.

Returns:

the name of the host connected to

See also:

getPort (p. 484)

4.121.3.3 const char * ArTcpConnection::getOpenMessage (int messageNumber) [virtual]

Gets the string of the message associated with opening the device.

Each class inherited from this one has an open method which returns 0 for success or an integer which can be passed into this function to obtain a string describing the reason for failure

Parameters:

messageNumber the number returned from the open

Returns:

the error description associated with the messageNumber

Reimplemented from ArDeviceConnection (p. 126).

4.121.3.4 int ArTcpConnection::getPort (void)

Gets the number of the port connected to.

Returns:

the number of the port connected to

See also:

getHost (p. 483)

4.121.3.5 int ArTcpConnection::getStatus (void) [virtual]

Gets the status of the connection, which is one of the enum status.

Gets the status of the connection, which is one of the enum status. If you want to get a string to go along with the number, use getStatusMessage

Returns:

the status of the connection

See also:

```
getStatusMessage (p. 127)
```

Reimplemented from **ArDeviceConnection** (p. 126).

4.121.3.6 ArTime ArTcpConnection::getTimeRead (int index) [virtual]

Gets the time data was read in.

Parameters:

index looks like this is the index back in the number of bytes last read in

Returns:

the time the last read data was read in

Reimplemented from ArDeviceConnection (p. 127).

4.121.3.7 bool ArTcpConnection::isTimeStamping (void) [virtual]

sees if timestamping is really going on or not.

Returns:

true if real timestamping is happening, false otherwise

Reimplemented from **ArDeviceConnection** (p. 127).

4.121.3.8 int ArTcpConnection::open (const char * host = NULL, int port = 8101)

Opens a connection to the given host and port.

Parameters:

```
host the host to connect to, if NULL (default) then localhost port the port to connect to
```

Returns:

0 for success, otherwise one of the open enums

See also:

getOpenMessage (p. 483)

4.121.3.9 int ArTcpConnection::read (const char * data, unsigned int size, unsigned int msWait = 0) [virtual]

Reads data from connection.

Reads data from connection

Parameters:

```
data pointer to a character array to read the data into size maximum number of bytes to read msWait read blocks for this many milliseconds (not at all for < 0)
```

Returns:

number of bytes read, or -1 for failure

See also:

```
write (p. 486), writePacket (p. 129)
```

Reimplemented from **ArDeviceConnection** (p. 128).

4.121.3.10 void ArTcpConnection::setSocket (ArSocket * socket)

Sets the tcp connection to use this socket instead of its own.

This will make the connection use this socket, its useful for doing funkier things with sockets but still being able to use a device connection.

Parameters:

sock the socket to use

4.121.3.11 int ArTcpConnection::write (const char * data, unsigned int size) [virtual]

Writes data to connection.

Writes data to connection

Parameters:

data pointer to a character array to write the data from
size number of bytes to write

Returns:

number of bytes read, or -1 for failure

See also:

read (p. 485), writePacket (p. 129)

Reimplemented from ArDeviceConnection (p. 128).

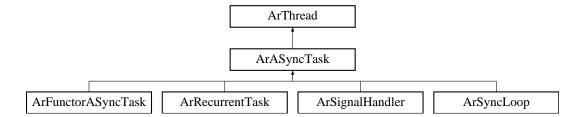
- \bullet ArTcpConnection.h
- ArTcpConnection.cpp

4.122 ArThread Class Reference

POSIX/WIN32 thread wrapper class.

#include <ArThread.h>

Inheritance diagram for ArThread::



Public Types

 enum Status { STATUS_FAILED = 1, STATUS_NORESOURCE, STATUS_NO_SUCH_THREAD, STATUS_INVALID, STATUS_ JOIN_SELF, STATUS_ALREADY_DETATCHED }

Public Methods

• ArThread (bool blockAllSignals=true)

Constructor.

• **ArThread** (ThreadType thread, bool joinable, bool blockAll-Signals=true)

Constructor - starts the thread.

• ArThread (ArFunctor *func, bool joinable=true, bool blockAll-Signals=true)

Constructor - starts the thread.

• virtual ~**ArThread** ()

Destructor.

• virtual int **create** (**ArFunctor** *func, bool joinable=true, bool lower-Priority=true)

Create and start the thread.

• virtual void **stopRunning** (void)

Stop the thread.

• virtual int **join** (void **ret=NULL)

Join on the thread.

• virtual int **detach** (void)

Detatch the thread so it cant be joined.

• virtual void cancel (void)

Cancel the thread.

• virtual bool **getRunning** (void) const

Get the running status of the thread.

• virtual bool **getRunningWithLock** (void)

Get the running status of the thread, locking around the variable.

• virtual bool getJoinable (void) const

Get the joinable status of the thread.

• virtual const ThreadType * **getThread** (void) const

Get the underlying thread type.

• virtual **ArFunctor** * **getFunc** (void) const

Get the functor that the thread runs.

• virtual void **setRunning** (bool running)

 $Set\ the\ running\ value\ on\ the\ thread.$

• int **lock** (void)

Lock the thread instance.

• int tryLock (void)

Try to lock the thread instance without blocking.

• int unlock (void)

Unlock the thread instance.

• bool **getBlockAllSignals** (void)

Do we block all process signals at startup?

Static Public Methods

• void init (void)

Initialize the internal book keeping structures.

• ArThread * self (void)

Returns the instance of your own thread.

• void **stopAll** ()

Stop all threads.

• void cancelAll (void)

Cancel all threads.

• void **joinAll** (void)

Join on all threads.

• void yieldProcessor (void)

Yield the processor to another thread.

Protected Attributes

• bool myRunning

State variable to denote when the thread should continue or exit.

4.122.1 Detailed Description

POSIX/WIN32 thread wrapper class.

create() (p. 487) will create the thread. That thread will run the given Functor.

A thread can either be in a detached state or a joinable state. If the thread is in a detached state, that thread can not be **join**() (p. 488)'ed upon. The thread will simply run until the program exits, or its function exits. A joinable thread means that another thread and call **join**() (p. 488) upon it. If this function is called, the caller will block until the thread exits its function. This gives a way to synchronize upon the lifespan of threads.

Calling cancel() (p. 488) will cancel the thread.

The static function **self()** (p. 490) will return a thread

4.122.2 Member Enumeration Documentation

4.122.2.1 enum ArThread::Status

Enumeration values:

STATUS_FAILED Failed to create the thread.

STATUS_NORESOURCE Not enough system resources to create the thread.

STATUS_NO_SUCH_THREAD The thread can no longer be found.

STATUS_INVALID Thread is detached or another thread is joining on it.

STATUS_JOIN_SELF Thread is your own thread. Can't join on self. STATUS_ALREADY_DETATCHED Thread is already detatched.

4.122.3 Member Function Documentation

4.122.3.1 void ArThread::init (void) [static]

Initialize the internal book keeping structures.

Initializes the internal structures which keep track of what thread is what. This is called by **Aria::init()** (p. 208), so the user will not normally need to call this function themselves. This function *must* be called from the main thread of the application. In otherwords, it should be called by main().

4.122.3.2 ArThread * ArThread::self (void) [static]

Returns the instance of your own thread.

If a newly created thread calls **self**() (p. 490) on itself too soon, this will return NULL. This is due to the fact that the thread is first created and started. Then the operating system returns the thread ID and thread that called **create**() (p. 487) then updates the list of threads with the new thread ID. There is just not much that can be done about that. The use should be aware of this caveat.

- ArThread.h
- ArThread.cpp
- ArThread_LIN.cpp
- ArThread_WIN.cpp

4.123 ArTime Class Reference

A class for time readings.

#include <ariaUtil.h>

Public Methods

• ArTime ()

Constructor.

• \sim ArTime ()

Destructor.

• long mSecSince (ArTime since) const

Gets the number of milliseconds since the given timestamp to this one.

• long secSince (ArTime since) const

Gets the number of seconds since the given timestamp to this one.

• long **mSecTo** (void) const

Finds the number of millisecs from when this timestamp is set to to now.

• long **secTo** (void) const

Finds the number of seconds from when this timestamp is set to to now.

• long mSecSince (void) const

Finds the number of milliseconds from this timestamp to now.

• long **secSince** (void) const

Finds the number of seconds from when this timestamp was set to now.

• bool **isBefore** (ArTime testTime) const

returns whether the given time is before this one or not.

• bool **isAt** (ArTime testTime) const

 $returns\ whether\ the\ given\ time\ is\ equal\ to\ this\ time\ or\ not.$

• bool **isAfter** (ArTime testTime) const

returns whether the given time is after this one or not.

• void **setToNow** (void)

Sets the time to now.

- void addMSec (long ms)

 Add some milliseconds (can be negative) to this time.
- void **setSec** (time_t sec)

 Sets the seconds since 1970.
- void **setMSec** (time_t msec)

 Sets the milliseconds.
- time_t getSec (void) const Gets the seconds since 1970.
- time_t **getMSec** (void) const Gets the milliseconds.
- void **log** (void) const

 Logs the time.

4.123.1 Detailed Description

A class for time readings.

This class is for getting the time of certain events. This class is not for generic time stuff, just for timeStamping, hence the only commands are very simple and the accessors for getting the data directly shouldn't really be used. DON'T use this for keeping track of what time it is, its just for relative timing (ie this loop needs to sleep another 100 ms);

The documentation for this class was generated from the following file:

• ariaUtil.h

4.124 ArTransform Class Reference

A class to handle transforms between different coordinates.

#include <ArTransform.h>

Public Methods

• ArTransform ()

Constructor.

• ArTransform (ArPose pose)

Constructor, Sets the transform so points in this coord system transform to abs world coords.

• ArTransform (ArPose pose1, ArPose pose2)

Constructor, sets the transform so that pose1 will be transformed to pose2.

• virtual ~ArTransform ()

Destructor.

• ArPose doTransform (ArPose source)

Take the source pose and run the transform on it to put it into abs coordinates.

• ArPoseWithTime doTransform (ArPoseWithTime source)

Take the source pose and run the transform on it to put it into abs coordinates.

• ArPose doInvTransform (ArPose source)

Take the source pose and run the inverse transform on it, taking it from abs coords to local.

• ArPoseWithTime doInvTransform (ArPoseWithTime source)

Take the source pose and run the inverse transform on it, taking it from abs coords to local.

• void **doTransform** (std::list< **ArPose** *> *poseList)

Take a std::list of sensor readings and do the transform on it.

• void **doTransform** (std::list< **ArPoseWithTime** *> *poseList)

Take a std::list of sensor readings and do the transform on it.

• void **setTransform** (**ArPose** pose)

Sets the transform so points in this coord system transform to abs world coords.

• void setTransform (ArPose pose1, ArPose pose2)

Sets the transform so that pose1 will be transformed to pose2.

• double **getTh** ()

Gets the transform angle value (degrees).

4.124.1 Detailed Description

A class to handle transforms between different coordinates.

4.124.2 Member Function Documentation

4.124.2.1 ArPoseWithTime ArTransform::doInvTransform (ArPoseWithTime source) [inline]

Take the source pose and run the inverse transform on it, taking it from abs coords to local.

The source and result can be the same

Parameters:

source the parameter to transform

Returns:

the source transformed from absolute into local coords

4.124.2.2 ArPose ArTransform::doInvTransform (ArPose source) [inline]

Take the source pose and run the inverse transform on it, taking it from abs coords to local.

The source and result can be the same

Parameters:

 $\boldsymbol{source}\,$ the parameter to transform

Returns

the source transformed from absolute into local coords

4.124.2.3 ArPoseWithTime ArTransform::doTransform (ArPoseWithTime source) [inline]

Take the source pose and run the transform on it to put it into abs coordinates.

Parameters:

source the parameter to transform

Returns:

the source transformed into absolute coordinates

4.124.2.4 ArPose ArTransform::doTransform (ArPose source) [inline]

Take the source pose and run the transform on it to put it into abs coordinates.

Parameters:

source the parameter to transform

Returns:

the source transformed into absolute coordinates

4.124.2.5 void ArTransform::setTransform (ArPose pose1, ArPose pose2)

Sets the transform so that pose1 will be transformed to pose2.

Parameters:

pose1 transform this into pose2
pose2 transform pose1 into this

4.124.2.6 void ArTransform::setTransform (ArPose pose)

Sets the transform so points in this coord system transform to abs world coords.

Parameters:

pose the coord system from which we transform to abs world coords

- ArTransform.h
- ArTransform.cpp

4.125 ArTypes Class Reference

Contains platform independent sized variable types.

#include <ariaTypedefs.h>

Public Types

- typedef char **Byte**A single signed byte.
- typedef short **Byte2**Two signed bytes.
- typedef int **Byte4**Four signed bytes.
- typedef unsigned char **UByte**A single unsigned byte.
- typedef unsigned short **UByte2**Two unsigned bytes.
- typedef unsigned int **UByte4**Four unsigned bytes.

4.125.1 Detailed Description

Contains platform independent sized variable types.

The documentation for this class was generated from the following file:

• ariaTypedefs.h

4.126 ArUtil Class Reference

This class has utility functions.

#include <ariaUtil.h>

Public Types

• enum BITS { BIT0 = 0x1, BIT1 = 0x2, BIT2 = 0x4, BIT3 = 0x8, BIT4 = 0x10, BIT5 = 0x20, BIT6 = 0x40, BIT7 = 0x80, BIT8 = 0x100, BIT9 = 0x200, BIT10 = 0x400, BIT11 = 0x800, BIT12 = 0x1000, BIT13 = 0x2000, BIT14 = 0x4000, BIT15 = 0x8000 }

Values for the bits from 0 to 16.

enum REGKEY { REGKEY_CLASSES_ROOT, REGKEY_CURRENT_USER, REGKEY_LOCAL_MACHINE, REGKEY_USERS }

Static Public Methods

- void **sleep** (unsigned int ms)

 Sleep for the given number of milliseconds.
- unsigned int **getTime** (void)

 Get the time in milliseconds.
- template<class T> void **deleteSet** (T begin, T end)

 Delete all members of a set. Does NOT empty the set.
- template<class T> void **deleteSetPairs** (T begin, T end)

 Delete all members of a set. Does NOT empty the set.
- void **splitString** (std::string inString, std::list< std::string > &outList) Split a string into a set of words.
- long **sizeFile** (std::string fileName)

 OS-independent way of finding the size of a file.
- bool **findFile** (const char *fileName)

 OS-independent way of checking to see if a file exists and is readable.
- bool **stripDir** (std::string fileIn, std::string &fileOut)

 OS-independent way of stripping the directory from the filename.

- bool **stripFile** (std::string fileIn, std::string &fileOut)

 OS-independent way of stripping the filename from the directory.
- void appendSlash (std::string &path)

 Appends a slash to a path if there is not one there already.
- void **fixSlashes** (std::string &path)

 Fix the slash orientation in file path string for windows or linux.
- void fixSlashesForward (std::string &path)
 Fix the slash orientation in file path string to be all forward.
- void fixSlashesBackward (std::string &path)

 Fix the slash orientation in file path string to be all backward.
- int **strcmp** (std::string str, std::string str2)

 Finds out if two strings are equal.
- int **strcmp** (std::string str, const char *str2)

 Finds out if two strings are equal.
- int **strcmp** (const char *str, std::string str2)

 Finds out if two strings are equal.
- int **strcmp** (const char *str, const char *str2)

 Finds out if two strings are equal.
- void **escapeSpaces** (char *dest, const char *src)

 Puts a \ before spaces in src, puts it into dest.
- std::string **getStringFromFile** (const char *fileName)

 Returns a string contained in an arbitrary file.
- bool **getStringFromRegistry** (**REGKEY** root, const char *key, const char *value, char *str, int len)

 Returns a string from the Windows registry.

4.126.1 Detailed Description

This class has utility functions.

4.126.2 Member Enumeration Documentation

4.126.2.1 enum ArUtil::BITS

Values for the bits from 0 to 16.

Enumeration values:

BIT0 value of BIT0.

BIT1 value of BIT1.

BIT2 value of BIT2.

BIT3 value of BIT3.

BIT4 value of BIT4.

BIT5 value of BIT5.

BIT6 value of BIT6.

BIT7 value of BIT7.

BIT8 value of BIT8.

BIT9 value of BIT9.

BIT10 value of BIT10.

BIT11 value of BIT11.

BIT12 value of BIT12.

BIT13 value of BIT13.

BIT14 value of BIT14.

BIT15 value of BIT15.

4.126.2.2 enum ArUtil::REGKEY

These are for passing into getStringFromRegistry

Enumeration values:

REGKEY_CLASSES_ROOT use HKEY_CLASSES_ROOT.

REGKEY_CURRENT_CONFIG use HKEY_CURRENT_CONFIG.

REGKEY_CURRENT_USER use HKEY_CURRENT_USER.

REGKEY_LOCAL_MACHINE use HKEY_LOCAL_MACHIE.

REGKEY_USERS use HKEY_USERS.

4.126.3 Member Function Documentation

4.126.3.1 void ArUtil::appendSlash (std::string & path) [static]

Appends a slash to a path if there is not one there already.

Parameters:

path the path to append a slash to

4.126.3.2 template < class T > void ArUtil::deleteSet (T begin, T end) [inline, static]

Delete all members of a set. Does NOT empty the set.

Assumes that T is an iterator that supports the operator *, operator!= and operator++. The return is assumed to be a pointer to a class that needs to be deleted.

4.126.3.3 template < class T > void ArUtil::deleteSetPairs (T begin, T end) [inline, static]

Delete all members of a set. Does NOT empty the set.

Assumes that T is an iterator that supports the operator **, operator!= and operator++. The return is assumed to be a pair. The second value of the pair is assumed to be a pointer to a class that needs to be deleted.

4.126.3.4 void ArUtil::escapeSpaces (char * dest, const char * src) [static]

Puts a \ before spaces in src, puts it into dest.

This copies src into dest but puts a \ before any spaces in src, escaping them... its mostly for use with **ArArgumentBuilder** (p. 106)... make sure you have enough space in the arrays that you're passing as dest... this allocates no memory

4.126.3.5 bool ArUtil::findFile (const char * fileName) [static]

OS-independent way of checking to see if a file exists and is readable.

Returns:

true if file is found

Parameters:

fileName name of the file to size

4.126.3.6 void ArUtil::fixSlashes (std::string & path) [static]

Fix the slash orientation in file path string for windows or linux.

Parameters:

path the path in which to fix the orientation of the slashes

4.126.3.7 void ArUtil::fixSlashesBackward (std::string & path) [static]

Fix the slash orientation in file path string to be all backward.

Parameters:

path the path in which to fix the orientation of the slashes

4.126.3.8 void ArUtil::fixSlashesForward (std::string & path) [static]

Fix the slash orientation in file path string to be all forward.

Parameters:

path the path in which to fix the orientation of the slashes

4.126.3.9 std::string ArUtil::getStringFromFile (const char * fileName) [static]

Returns a string contained in an arbitrary file.

This function looks in the given filename and extracts a string from the file. The string can contain spaces or tabs, but a '\r' or '

' will be treated as the end of the string, and the string cannot have more than 1024 characters. This is mostly for use with Linux to pick up the **Aria** (p. 205) directory from a file in /etc, but will work with Linux or Windows.

Parameters:

filename the filename to look in

Returns:

the string that was in the file, or a string with length 0 if the file was not found or if the file was empty

4.126.3.10 bool ArUtil::getStringFromRegistry (REGKEY root, const char * key, const char * value, char * str, int len) [static]

Returns a string from the Windows registry.

This takes a root key, and looks up the given <key> within that root, then finds the string given to <value> and returns it.

Parameters:

root the root key to use, one of the REGKEY enums

key the name of the key to find

value the value to find the string contained in

str where to put the string sought, or if it could not be found for some reason an empty (length() == 0) string

len the length of the allocated memory in str

Returns:

true if the string was found, false if it was not found or if there was a problem such as the string not being long enough

4.126.3.11 unsigned int ArUtil::getTime (void) [static]

Get the time in milliseconds.

Get the time in milliseconds, counting from some arbitrary point. This time is only valid within this run of the program.

Returns:

millisecond time

4.126.3.12 long ArUtil::sizeFile (std::string fileName) [static]

OS-independent way of finding the size of a file.

Returns:

size in bytes. -1 on error.

Parameters:

fileName name of the file to size

4.126.3.13 void ArUtil::sleep (unsigned int ms) [static]

Sleep for the given number of milliseconds.

This sleeps for the given number of milliseconds... Note in linux it tries to sleep for 10 ms less than the amount given, which should wind up close to correct... Linux is broken in this regard and sleeps for too long... it sleeps for the ceiling of the current 10 ms range, then for an additional 10 ms... so: 11 to 20 ms sleeps for 30 ms... 21 to 30 ms sleeps for 40 ms... 31 to 40 ms sleeps for 50 ms... this continues on up to the values we care about of.. 81 to 90 ms sleeps for 100 ms... 91 to 100 ms sleeps for 110 ms... so we'll sleep for 10 ms less than we want to, which should put us about right... guh

Parameters:

ms the number of milliseconds to sleep for

4.126.3.14 void ArUtil::splitString (std::string inString, std::list< std::string $> \& \ outList$) [static]

Split a string into a set of words.

Takes a string and splits it into a list of words. It appends the words to the outList. If there is nothing found, it will not touch the outList.

Parameters:

inString the input string to split
ourList the list in which to store the words that are found

4.126.3.15 int ArUtil::strcmp (const char * str, const char * str2) [static]

Finds out if two strings are equal.

This compares two strings, it returns an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

Parameters:

str the string to compare str2 the second string to compare

Returns:

an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

4.126.3.16 int ArUtil::strcmp (const char * str, std::string str2) [static]

Finds out if two strings are equal.

This compares two strings, it returns an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

Parameters:

str the string to compare str2 the second string to compare

Returns:

an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

4.126.3.17 int ArUtil::strcmp (std::string str, const char * str2) [static]

Finds out if two strings are equal.

This compares two strings, it returns an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

Parameters:

str the string to compare str2 the second string to compare

Returns:

an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

4.126.3.18 int ArUtil::strcmp (std::string str, std::string str2) [static]

Finds out if two strings are equal.

This compares two strings, it returns an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

Parameters:

str the string to compare

str2 the second string to compare

Returns:

an integer less than, equal to, or greater than zero if str is found, respectively, to be less than, to match, or be greater than str2.

4.126.3.19 bool ArUtil::stripDir (std::string fileIn, std::string & fileOut) [static]

OS-independent way of stripping the directory from the filename.

Works for \setminus and /. Returns true if something was actually done. Sets fileOut to be what ever the answer is.

Returns:

true if the path contains a file

Parameters:

fileIn input path/filename
fileOut output filename

4.126.3.20 bool ArUtil::stripFile (std::string fileIn, std::string & fileOut) [static]

OS-independent way of stripping the filename from the directory.

Works for \backslash and /. Returns true if something was actually done. Sets fileOut to be what ever the answer is.

Returns:

true if the file contains a path

Parameters:

fileIn input path/filename
fileOut output path

The documentation for this class was generated from the following files:

- \bullet ariaUtil.h
- ariaUtil.cpp

4.127 ArVCC4 Class Reference

Driver for the VCC4.

#include <ArVCC4.h>

Inheritance diagram for ArVCC4::



Public Methods

- ArVCC4 (ArRobot *robot, bool inverted=false, CommState comm-Direction=COMM_UNKNOWN, bool autoUpdate=true)
 - Constructor.
- virtual ~ArVCC4 ()

Destructor.

- virtual bool **init** (void)
 - Initializes the camera.
- bool **isInitted** (void)

Returns true if the camera has been initialized.

- virtual void connectHandler (void)
 - $Internal,\ attached\ to\ robot,\ inits\ the\ camera\ when\ robot\ connects.$
- virtual bool packetHandler (ArBasePacket *packet)

Handles a packet that was read from the device.

- virtual bool **pan** (int deg)
 - Pans to the given degrees.
- virtual bool panRel (int deg)

Pans relative to current position by given degrees.

• virtual bool tilt (int deg)

Tilts to the given degrees.

• virtual bool **tiltRel** (int deg)

Tilts relative to the current position by given degrees.

• virtual bool panTiltRel (int pdeg, int tdeg)

Pans and tilts relatives to the current position by the given degrees.

• virtual int getMaxPosPan (void) const

Gets the highest positive degree the camera can pan to.

• virtual int **getMaxNegPan** (void) const

Gets the lowest negative degree the camera can pan to.

• virtual int **getMaxPosTilt** (void) const

Gets the highest positive degree the camera can tilt to.

• virtual int getMaxNegTilt (void) const

Gets the lowest negative degree the camera can tilt to.

• void getRealPanTilt (void)

Requests that a packet be sent to the camera to retrieve what the camera thinks are its pan/tilt positions.

• void **getRealZoomPos** (void)

Requests that a packet be sent to the camera to retrieve what the camera thinks is its zoom position.

• virtual bool canZoom (void) const

Returns true if camera can zoom (or rather, if it is controlled by this).

• virtual bool **panTilt** (int pdeg, int tdeg)

Pans and tilts to the given degrees.

• virtual bool **zoom** (int deg)

Zooms to the given value.

• void addErrorCB (ArFunctor *functor, ArListPos::Pos position)

Adds an error callback to a list of callbacks to be called when there is a serious error in communicating - either the parameters were incorrect, the mode was incorrect, or there was an unknown error.

• void remErrorCB (ArFunctor *functor)

Remove an error callback from the callback list.

• bool haltPanTilt (void)

Halts all pan-tilt movement.

• bool haltZoom (void)

Halts zoom movement.

• bool **panSlew** (int deg)

Sets the rate that the unit pans at.

• bool tiltSlew (int deg)

Sets the rate the unit tilts at.

• void **preparePacket** (ArVCC4Packet *packet)

Adds device ID and delimeter to packet buffer.

• virtual int **getPan** (void) const

The angle the camera was last told to pan to.

• virtual int **getTilt** (void) const

The angle the camera was last told to tilt to.

• virtual int **getZoom** (void) const

The value the camera was last told to zoom to.

• int **getPanSlew** (void)

 $Gets\ the\ current\ pan\ slew.$

• int **getMaxPanSlew** (void)

Gets the maximum pan slew.

• int getMinPanSlew (void)

Gets the minimum pan slew.

• int **getTiltSlew** (void)

Gets the current tilt slew.

• int **getMaxTiltSlew** (void)

Gets the maximum tilt slew.

• int **getMinTiltSlew** (void)

Gets the minimum tilt slew.

- virtual int **getMaxZoom** (void) const
 - Gets the maximum value for the zoom on this camera.
- virtual int **getMinZoom** (void) const
 - Gets the lowest value for the zoom on this camera.
- bool wasError (void)

Returns true if the error callback list was called during the last cycle.

Protected Types

• enum Error { CAM_ERROR_NONE = 0x30, CAM_ERROR_-BUSY = 0x31, CAM_ERROR_PARAM = 0x35, CAM_ERROR_-MODE = 0x39, CAM_ERROR_UNKNOWN = 0xFF }

Protected Methods

• virtual ArBasePacket * readPacket (void)

Reads a packet from the device connection, MUST NOT BLOCK.

4.127.1 Detailed Description

Driver for the VCC4.

4.127.2 Member Enumeration Documentation

4.127.2.1 enum ArVCC4::Error [protected]

Enumeration values:

CAM_ERROR_NONE No error.

CAM_ERROR_BUSY Camera busy, will not execute the command.

CAM_ERROR_PARAM Illegal parameters to function call.

CAM_ERROR_MODE Not in host control mode.

CAM_ERROR_UNKNOWN Unknown error condition. Should never happen.

4.127.3 Constructor & Destructor Documentation

4.127.3.1 ArVCC4::ArVCC4 (ArRobot * robot, bool inverted = false, CommState commDirection = COMM_UNKNOWN, bool autoUpdate = true)

Constructor.

Parameters:

robot the robot this camera is attached to

inverted if this camera is inverted or not, the only time a camera will normally be inverted is on a robot where it's mounted on the underside of something, ie like in a peoplebot

commDirection this is the type of communications that the camera should use. It can be unidirectional, bidirectional, or unknown. If unidirectional it sends packets without knowing if the camera has received them or not. This results in necessary 300 ms delays between packets, otherwise the packets will get dropped. In bidirectional mode, responses are received from the camera and evaluated for success of receipt of the previous command. In unknown mode, it will use bidirectional communication if a response is received, otherwise it will be unidirectional.

cameraPort this is microcontroller port to use, if the camera is plugged into the microcontroller. Use 1 for Aux1, and 2 for Aux2. Use 0 if using a computer serial port such as Com2. If set to 0, then the usertask will not do anything until the device connection has been set with myVCC4->setDeviceConnection(ArDeviceConnection*)

auto Upate this will cause the usertask to periodically query the camera for actual positional information (pan, tilt, zoom). This will happen every 1 sec idle time, and will switch between pan/tilt info and zoom info.

4.127.4 Member Function Documentation

4.127.4.1 bool ArVCC4::packetHandler (ArBasePacket * packet) [virtual]

Handles a packet that was read from the device.

This should work for the robot packet handler or for packets read in from read-Packet (the joys of OO), but it can't deal with the need to check the id on robot packets, so you should check the id from robotPacketHandler and then call this one so that your stuff can be used by both robot and serial port connections.

Parameters:

packet the packet to handle

Returns:

true if this packet was handled (ie this knows what it is), false otherwise

Reimplemented from ArPTZ (p. 302).

4.127.4.2 ArBasePacket * ArVCC4::readPacket (void) [protected, virtual]

Reads a packet from the device connection, MUST NOT BLOCK.

This should read in a packet from the myConn connection and return a pointer to a packet if there was on to read in, or NULL if there wasn't one... this MUST not block if it is used with the default mode of being driven from the sensorInterpHandler, since that is on the robot loop.

Returns:

packet read in, or NULL if there was no packet read

Reimplemented from ArPTZ (p. 302).

The documentation for this class was generated from the following files:

- ArVCC4.h
- ArVCC4.cpp

4.128 ArVCC4Commands Class Reference

A class with the commands for the VCC4.

#include <ArVCC4.h>

Public Types

• enum Command { DELIM = 0x00, DEVICEID = 0x30, PANSLEW = 0x50, TILTSLEW = 0x51, STOP = 0x53, INIT = 0x58, SLEWREQ = 0x59, ANGLEREQ = 0x5c, PANTILT = 0x62, SETRANGE = 0x64, PANTILTREQ = 0x63, CONTROL = 0x90, POWER = 0xA0, ZOOMSTOP = 0xA2, ZOOM = 0xB3, FOOTER = 0xEF, RESPONSE = 0xFE, HEADER = 0xFF }

4.128.1 Detailed Description

A class with the commands for the VCC4.

This class is for controlling the Canon VC-C4 camera.

This camera has a reponse mechanism, whereby each packet sent to the camera generates an answer within 300ms. For the most part, the answer consists of a 6-byte packet which has an error-status within it. Some commands generate longer packets.

In order for the the reponses to work, the CTS line on the camera must be high. This is pin 2 on the visca port. If your camera is not wired in such a fashion, then no answers will be sent to the computer, and the computer will not know whether or not the last packet was processed correctly. Because of this, systems operating without the answer feature will need to run with delays between sending packets. Otherwise, packets will be ignored, but you will have no way of knowing that. To achieve this, there are two types of communication modes that this class will operate under - COMM_UNIDIRECTIONAL or COMM_BIDIRECTIONAL. The default is COMM_UNKNOWN, in which it will use bidirectional communication if a response is received.

To handle the states and packet processing, this class runs as a user-task, different than the other pan/tilt devices. Because of this, it must have a valid robot connection and a valid serial connection if using a computer serial port. Note that the computer port must be set independently of this class. The aux port can be selected via setAuxPort from the **ArPTZ** (p. 298) class.

The camera's pan and tilt commands work on a number of units equal to (degrees / 0.1125). The panTilt function always rounds the conversion closer to zero, so that a magnitude greater than the allowable range of movement is not sent to the camera.

4.128.2 Member Enumeration Documentation

4.128.2.1 enum ArVCC4Commands::Command

Enumeration values:

DELIM Delimeter character.

DEVICEID Default device ID.

PANSLEW Sets the pan slew.

TILTSLEW Sets the tilt slew.

STOP Stops current pan/tilt motion.

INIT Initializes the camera.

SLEWREQ Request pan/tilt min/max slew.

ANGLEREQ Request pan/tilt min/max angle.

PANTILT Pan/tilt command.

SETRANGE Pan/tilt min/max range assignment.

PANTILTREQ Request pan/tilt position.

CONTROL Puts camera in Control mode.

POWER Turns on/off power.

ZOOMSTOP Stops zoom motion.

ZOOM Zooms camera lens.

FOOTER Packet Footer.

RESPONSE Packet header for response.

HEADER Packet Header.

The documentation for this class was generated from the following file:

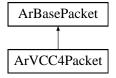
• ArVCC4.h

4.129 ArVCC4Packet Class Reference

A class for for making commands to send to the VCC4.

#include <ArVCC4.h>

Inheritance diagram for ArVCC4Packet::



Public Methods

- ArVCC4Packet (ArTypes::UByte2 bufferSize=30) Constructor.
- virtual \sim **ArVCC4Packet** () *Destructor.*
- virtual void **finalizePacket** (void)

MakeFinals the packet in preparation for sending, must be done.

4.129.1 Detailed Description

A class for for making commands to send to the VCC4.

There are only a few functioning ways to put things into this packet, you MUST use thse, if you use anything else your commands won't work. You must use byteToBuf and byte2ToBuf.

The documentation for this class was generated from the following files:

- \bullet ArVCC4.h
- ArVCC4.cpp

4.130 P2ArmJoint Class Reference

P2 Arm joint info.

#include <ArP2Arm.h>

4.130.1 Detailed Description

P2 Arm joint info.

The documentation for this class was generated from the following files:

- ArP2Arm.h
- \bullet ArP2Arm.cpp

Index

 \sim ArACTSBlob ArActionLimiterTableSensor, ArACTSBlob, 94 \sim ArActionStallRecover \sim ArACTS_1_2 ArACTS_1_2, 89 ArActionStallRecover, 83 \sim ArAMPTU ~ArActionStop Aramptu, 96 ArActionStop, 85 \sim ArAMPTUPacket \sim ArActionTurn ArAMPTUPacket, 101 ArActionTurn, 87 \sim ArASyncTask \sim ArArg ArArg, 103 ArASyncTask, 110 \sim ArAction \sim ArArgumentBuilder ArArgumentBuilder, 106 ArAction, 40 \sim ArActionAvoidFront \sim ArArgumentParser ArActionAvoidFront, 43 ArArgumentParser, 107 ~ArActionAvoidSide \sim ArBasePacket ArBasePacket, 112 ArActionAvoidSide, 45 ~ArActionBumpers \sim ArCondition ArCondition, 122 ArActionBumpers, 47 \sim ArDPPTU ~ArActionConstantVelocity ArActionConstantVelocity, 49 ArDPPTU, 130 \sim ArDPPTUPacket ~ArActionDesired ArDPPTUPacket, 138 ArActionDesired, 51 ~ArActionGroup \sim ArDeviceConnection ArDeviceConnection, 124 ArActionGroup, 60 \sim ArFunctor ~ArActionInput ArFunctor, 139 ArActionInput, 68 \sim ArFunctor1 ~ArActionJoydrive ArActionJoydrive, 70 ArFunctor1, 142 \sim ArFunctor1C ~ArActionKeydrive ArFunctor1C, 144 ArActionKeydrive, 74 \sim ArActionLimiterBackwards \sim ArFunctor2 ArActionLimiterBackwards, ArFunctor2, 148 77 \sim ArFunctor2C \sim ArActionLimiterForwards ArFunctor2C, 150 ArActionLimiterForwards, 79 \sim ArFunctor3 \sim ArActionLimiterTableSensor ArFunctor3, 155

~ArFunctor3C \sim ArModeTeleop ArFunctor3C, 158 ArModeTeleop, 254 \sim ArFunctorASyncTask ~ArModeUnguardedTeleop ArFunctorASyncTask, 164 ArModeUnguardedTeleop, 256 \sim ArFunctorC \sim ArModeWander ArFunctorC, 165 ArModeWander, 258 \sim ArGlobalFunctor \sim ArModule ArGlobalFunctor, 168 ArModule, 260 \sim ArGlobalFunctor1 ${\sim} ArMutex$ ArGlobalFunctor1, 170 ArMutex, 266 ~ArGlobalFunctor2 \sim ArNetServer ArGlobalFunctor2, 173 ArNetServer, 268 \sim ArGlobalFunctor3 ~ArNetServerConnection ArGlobalFunctor3, 177 ArNetServerConnection, 271 \sim ArGlobalRetFunctor \sim ArP2Arm ArP2Arm, 273 ArGlobalRetFunctor, 182 \sim ArGlobalRetFunctor1 \sim ArPTZ ArGlobalRetFunctor1, 184 ArPTZ, 298 \sim ArGlobalRetFunctor2 \sim ArPose ArGlobalRetFunctor2, 187 ArPose, 285 ~ArGlobalRetFunctor3 \sim ArPref ArGlobalRetFunctor3, 191 ArPref, 290 \sim ArGripper ~ArPriorityResolver ArGripper, 196 ArPriorityResolver, 297 \sim ArInterpolation \sim ArRangeBuffer ArInterpolation, 210 ArRangeBuffer, 305 \sim ArIrrfDevice \sim ArRangeDevice ArIrrfDevice, 212 ArRangeDevice, 312 \sim ArJoyHandler \sim ArRangeDeviceThreaded ArJoyHandler, 214 ArRangeDeviceThreaded, 320 \sim ArKeyHandler \sim ArRecurrentTask ArKeyHandler, 220 ArRecurrentTask, 323 \sim ArLine \sim ArResolver ArLine, 224 ArResolver, 325 \sim ArLineSegment \sim ArRetFunctor ArLineSegment, 226 ArRetFunctor, 327 \sim ArLogFileConnection ~ArRetFunctor1 ArLogFileConnection, 233 ArRetFunctor1, 328 \sim ArMode ~ArRetFunctor1C ArRetFunctor1C, 330 ArMode, 244 \sim ArModeCamera ${\sim} ArRetFunctor2$ ArModeCamera, 248 ArRetFunctor2, 334 \sim ArModeGripper ~ArRetFunctor2C ArModeGripper, 250 ArRetFunctor2C, 337 \sim ArModeSonar ~ArRetFunctor3 ArModeSonar, 252 ArRetFunctor3, 342

\sim ArRetFunctor3C	\sim ArVCC4Packet
ArRetFunctor3C, 346	ArVCC4Packet, 514
~ArRetFunctorC	Tii v O O II diolect, O I I
ArRetFunctorC, 352	ABSPAN
~ArRobot	ArAMPTUCommands, 99
ArRobot, 355	ABSTILT
~ArRobotPacket	ArAMPTUCommands, 99
ArRobotPacket, 406	ACCEL
~ArRobotPacketReceiver	ArDPPTUCommands, 136
ArRobotPacketReceiver, 408	accept
~ArRobotPacketSender	ArSocket, 462
ArRobotPacketSender, 411	accountForRobotHeading
~ArRobotParams	ArActionDesired, 54
ArRobotParams, 415	actionHandler
~ArSectors	ArRobot, 371
ArSectors, 418	ActionMap
~ArSerialConnection	ArResolver, 325
	activate
ArSerialConnection, 424 ~ArSick	ArAction, 40
	ArActionGroup, 60
ArSick, 431	ArActionInput, 68
~ArSickLogger	ArActionHiput, 08 ArActionKeydrive, 74
ArSickLogger, 444	ArMode, 245
~ArSickPacket	
ArSickPacket, 447	ArModeCamera, 248
~ArSickPacketReceiver	ArModeGripper, 250
ArSickPacketReceiver, 450	ArModeSonar, 252
~ArSignalHandler	ArModeTeleop, 254
ArSignalHandler, 453	ArModeUnguardedTeleop, 256
~ArSimpleConnector	ArModeWander, 258
ArSimpleConnector, 459	activateExclusive
~ArSocket	ArActionGroup, 60
ArSocket, 461	ACTIVE
~ArSonarDevice	ArTaskState, 480
ArSonarDevice, 468	ActsConstants
~ArSyncTask	ArACTS_1_2, 90
ArSyncTask, 477	actsHandler
~ArTcpConnection	ArACTS_1_2, 90
ArTcpConnection, 481	add
~ArThread	ArArgumentBuilder, 106
ArThread, 487	addAction
~ArTime	ArActionGroup, 61
ArTime, 491	ArRobot, 371
~ArTransform	addAngle
ArTransform, 493	ArMath, 239
~ArVCC4	addAverage
ArVCC4, 506	ArActionDesired, 54

addCommand	ArPobot 274
addCommand	ArRobot, 374
ArNetServer, 269	addSensorInterpTask
addConnectCB	ArRobot, 374
ArRobot, 372	addTagToLog
ArSick, 437	ArSickLogger, 445
addDisconnectNormallyCB	addTagToLogPlain
ArRobot, 372	ArSickLogger, 444
ArSick, 438	addUninitCallBack
addDisconnectOnErrorCB	Aria, 207
ArRobot, 372	addUserTask
ArSick, 438	ArRobot, 375
addErrorCB	ADSEL
ArVCC4, 507	ArCommands, 120
addFailedConnectCB	ALREADY_CONNECTED
ArRobot, 373	ArP2Arm, 278
ArSick, 439	ALREADY_INITED
addGoal	ArP2Arm, 277
ArSickLogger, 444	angleBetween
addHandlerCB	ArMath, 238
ArSignalHandler, 455	ANGLEREQ
addInitCallBack	ArVCC4Commands, 513
Aria, 206	appendSlash
addKeyHandler	ArUtil, 500
ArKeyHandler, 222	apply Transform
addMSec	ArRangeBuffer, 307
ArTime, 492	ArRangeDevice, 315
addNewBranch	ArRobot, 375
ArSyncTask, 477	ArSensorReading, 421
addNewLeaf	ArAction
ArSyncTask, 477	\sim ArAction, 40
addPacketHandler	activate, 40
ArRobot, 373	ArAction, 40
addPlain	deactivate, 41
ArArgumentBuilder, 106	getArg, 41
addRangeDevice	getDescription, 41
ArRobot, 363	getDesired, 41
addReading	getName, 41
ArInterpolation, 210	getNumArgs, 41
ArRangeBuffer, 307	isActive, 40
ArRangeDevice, 312	log, 41
ArSonarDevice, 469	setNextArgument, 41
addrHost	setRobot, 41
ArSocket, 464	ArAction, 39
addRobot	fire, 42
Aria, 205	ArActionAvoidFront
addRunExitCB	~ArActionAvoidFront, 43
	-, -

ArActionAvoidFront, 44	getMaxVelStrength, 52
getDesired, 43	getVel, 51
ArActionAvoidFront, 43	getVelDesiredChannel, 53
ArActionAvoidFront, 44	getVelStrength, 52
fire, 44	reset, 51
ArActionAvoidSide	ArActionDesired, 51
\sim ArActionAvoidSide, 45	accountForRobotHeading, 54
ArActionAvoidSide, 45	addAverage, 54
getDesired, 45	endAverage, 54
ArActionAvoidSide, 45	merge, 54
ArActionAvoidSide, 45	setDeltaHeading, 54
fire, 46	setHeading, 55
ArActionBumpers	setMaxNegVel, 55
\sim ArActionBumpers, 47	setMaxRotVel, 55
ArActionBumpers, 48	setMaxVel, 55
getDesired, 47	setVel, 56
ArActionBumpers, 47	startAverage, 56
ArActionBumpers, 48	ArActionDesiredChannel, 57
fire, 48	MAX_STRENGTH, 57
ArActionConstantVelocity	MIN_STRENGTH, 57
\sim ArActionConstantVelocity,	NO_STRENGTH, 57
49	ArActionGoto
ArActionConstantVelocity, 49	cancelGoal, 58
getDesired, 49	getCloseDist, 58
ArActionConstantVelocity, 49	getDesired, 59
ArActionConstantVelocity, 49	getGoal, 58
fire, 50	getSpeed, 58
ArActionDesired	haveAchievedGoal, 58
\sim ArActionDesired, 51	setCloseDist, 58
ArActionDesired, 51	setGoal, 58
getDeltaHeading, 52	setSpeed, 58
getDeltaHeadingDesired-	ArActionGoto, 58
Channel, 53	fire, 59
getDeltaHeadingStrength, 52	ArActionGroup
getHeading, 52	~ArActionGroup, 60
getHeadingStrength, 52	activate, 60
getMaxNegVel, 52	activateExclusive, 60
getMaxNegVelDesired-	ArActionGroup, 61
Channel, 53	deactivate, 60
getMaxNegVelStrength, 52	getActionList, 60
getMaxRotVel, 52	removeActions, 60
getMaxRotVelDesiredChannel,	ArActionGroup, 60
53	addAction, 61
getMaxRotVelStrength, 52	ArActionGroup, 61
getMaxVel, 52	remAction, 61
getMaxVelDesiredChannel, 53	ArActionGroupInput, 63

ArActionGroupUnguardedTeleop, 66 ArActionGroupWander, 67 ArActionInput, 68 activate, 68 artivate, 68 deltaHeading, 68 deltaHeading fromCurrent, 68 deltaHeading from Current, 68 fre, 78 ArActionLimiterBackwards, 77 ArActionLimiterForwards, 80 fi	ArActionGroupStop, 64	takeKeys, 75
ArActionGroupUnguardedTeleop, 66 ArActionGroupWander, 67 ArActionInput	·	
ArActionGroupWander, 67 ArActionInput		
$ \begin{array}{llllllllllllllllllllllllllllllllllll$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		
~ArActionInput, 68 activate, 68 ArActionInput, 69 deltaHeading, 68 deltaHeading, 68 deltaHeading, 68 deltaVel, 68 getDesired, 68 setVel, 68 ArActionInput, 69 fire, 69 ArActionJoydrive ~ArActionJoydrive, 70 ArActionJoydrive, 71 getDesired, 71 getStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setThrottleParams, 70 ArActionJoydrive, 70 ArActionStallRecover, 81 ArActionStallRecover, 83 ArActionStallRecover, 83 ArActionStallRecover, 84 getDesired, 83 ArActionStallRecover, 84 getDesired, 85 ArActionStop, 85 getDesired, 85 ArActionStop, 85 getDesired, 85 ArActionStop, 85 fire, 86 ArActionLimiterBackwards, 76 ArActionLimiterBackwards, 77 ArActionLimiterForwards, 80 getDesired, 79 ArActionLimiterTableSensor ArActionLimiterForwards, 80 getDesired, 79		
activate, 68 ArActionInput, 69 deltaHeading, 68 deltaHeadingFromCurrent, 68 deltaVel, 68 getDesired, 68 getDesired, 68 setVel, 68 ArActionInput, 69 fire, 69 ArActionJoydrive ArActionJoydrive, 70 ArActionJoydrive, 71 getDesired, 71 getStopIfNoButtonPressed, 70 joystickInited, 70 setSpeeds, 70 setThrottleParams, 70 ArActionJoydrive, 71 fire, 72 getUseOSCal, 72 setUseOSCal, 72 setUseOSCal, 72 setUseOSCal, 72 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 ArActionTimiterFackwards, 77 ArActionLimiterFackwards, 77 ArActionLimiterForwards, 79 ArActionLimiterForwards, 80 getDesired, 81 ArActionLimiterForwards, 80 getDesired, 81 ArActionLimiterForwards, 80 getDesired, 79 ArActionLimiterForwards, 80 getDesired,		
ArActionInput, 69 deltaHeading, 68 deltaHeadingFromCurrent, 68 deltaVel, 68 getDesired, 68 setVel, 68 ArActionInput, 69 fire, 69 ArActionJoydrive		
deltaHeading, 68 deltaHeadingFromCurrent, 68 deltaVel, 68 getDesired, 68 getDesired, 68 setVel, 68 ArActionInput, 68 ArActionInput, 69 fire, 69 ArActionJoydrive		
deltaHeadingFromCurrent, 68 deltaVel, 68 getDesired, 68 setVel, 68 ArActionInput, 68 ArActionInput, 69 fire, 69 ArActionJoydrive ArActionJoydrive, 70 ArActionJoydrive, 71 getDesired, 71 getStopIfNoButtonPressed, 70 joystickInited, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 ArActionJoydrive, 70 ArActionStallRecover, 81 fire, 72 getUseOSCal, 72 setUseOSCal, 72 setUseOSCal, 72 ArActionKeydrive ArActionKeydrive ArActionKeydrive ArActionKeydrive, 74 deactivate, 74 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 ArActionTurn ArActionTurimiterForwards, 80 getDesired, 79 ArActionLimiterForwards, 80 getDesired, 79 ArActionLimiterForwards, 80 getDesired, 80 getDesired, 81 ArActionLimiterTableSensor ArActionLimite		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	<u> </u>	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		· · · · · · · · · · · · · · · · · · ·
set Vel, 68 ArActionInput, 68 ArActionInput, 69 fire, 69 ArActionJoydrive ArActionJoydrive, 70 ArActionJoydrive, 71 getDesired, 71 getJoyHandler, 71 getStopIfNoButtonPressed, 70 joystickInited, 70 setSpeeds, 70 ArActionJoydrive, 70 ArActionJoydrive, 70 ArActionJoydrive, 70 setStopIfNoButtonPressed, 70 setArctionLimiterTableSensor ArActionLimiterTableSensor, 81 fire, 81 ArActionLimiterTableSensor, 81 fire, 82 ArActionLimiterTableSensor ArActionLimiterTableSensor, 81 fire, 82 ArActionLimiterTableSensor ArActionLimiterTableSensor ArActionLimiterTableSensor ArActionLimiterTableSensor ArActionLimiterForwards, 80 getDesired, 79 ArActionLimiterForwards, 70 ArActionLimiterForwards, 80 getDesired, 79 ArActionLimiterForwards, 80 fire, 80 ArActionLimiterForwards, 80 getDesired, 79 ArActionLimiterForwards, 80 fire, 80 ArActionLimiterForwards, 79 ArActionLimiterForwards, 79 ArActionLimit	· · · · · · · · · · · · · · · · · · ·	
$\begin{array}{llllllllllllllllllllllllllllllllllll$	-	
ArActionInput, 69 fire, 69 ArActionJoydrive		,
$\begin{array}{lll} & 79 \\ \text{ArActionJoydrive} \\ \sim & \text{ArActionJoydrive}, 70 \\ \text{ArActionJoydrive}, 71 \\ \text{getDesired}, 71 \\ \text{getJoyHandler}, 71 \\ \text{getStopIfNoButtonPressed}, 70 \\ \text{joystickInited}, 70 \\ \text{setSpeeds}, 70 \\ \text{setStopIfNoButtonPressed}, 70 \\ \text{setThrottleParams}, 70 \\ \text{ArActionJoydrive}, 70 \\ \text{ArActionJoydrive}, 71 \\ \text{fire}, 72 \\ \text{getUseOSCal}, 72 \\ \text{setUseOSCal}, 72 \\ \text{setUseOSCal}, 72 \\ \text{ArActionKeydrive} \\ \sim & \text{ArActionKeydrive} \\ \text{ArActionKeydrive}, 74 \\ \text{activate}, 74 \\ \text{ActionKeydrive}, 74 \\ \text{deactivate}, 74 \\ \text{down}, 75 \\ \text{getDesired}, 74 \\ \text{giveUpKeys}, 75 \\ \text{left}, 75 \\ \text{right}, 75 \\ \text{setIncrements}, 74 \\ \text{setSpeeds}, 74 \\ \end{array} \begin{array}{l} 79 \\ \text{ArActionLimiterForwards}, 80 \\ \text{getDesired}, 79 \\ \text{ArActionLimiterTableSensor} \\ \text{ArActionLimiterTableSensor}, 81 \\ \text{ArActionLimiterTableSensor}, 81 \\ \text{fire}, 82 \\ \text{ArActionStallRecover} \\ \text{ArActionStallRecover}, 83 \\ \text{ArActionStallRecover}, 84 \\ \text{getDesired}, 83 \\ \text{ArActionStallRecover}, 84 \\ \text{fire}, 84 \\ \text{ArActionStop}, 85 \\ \text{ArActionStop}, 85 \\ \text{getDesired}, 85 \\ \text{ArActionStop}, 85 \\ \text{setIncrements}, 74 \\ \text{setRobot}, 74 \\ \text{setSpeeds}, 74 \\ \end{array}$		
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	- · ·	
~ArActionJoydrive, 70 ArActionJoydrive, 71 getDesired, 71 getDesired, 71 getJoyHandler, 71 getStopIfNoButtonPressed, 70 joystickInited, 70 setSpeeds, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setThrottleParams, 70 ArActionJoydrive, 70 ArActionJoydrive, 70 ArActionJoydrive, 71 fire, 72 getUseOSCal, 72 setUseOSCal, 72 ArActionKeydrive ~ArActionKeydrive		
ArActionJoydrive, 71 getDesired, 71 getJoyHandler, 71 getStopIfNoButtonPressed, 70 joystickInited, 70 setSpeeds, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setThrottleParams, 70 ArActionJoydrive, 70 ArActionJoydrive, 71 fire, 72 getUseOSCal, 72 ArActionKeydrive	· ·	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	* '	
getJoyHandler, 71 getStopIfNoButtonPressed, 70 joystickInited, 70 setSpeeds, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setStopIfNoButtonPressed, 70 setThrottleParams, 70 ArActionJoydrive, 70 ArActionJoydrive, 71 fire, 72 getUseOSCal, 72 setUseOSCal, 72 ArActionKeydrive ~ArActionKeydrive ~ArActionKeydrive, 74 activate, 74 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 fire, 80 ArActionLimiterTableSensor, 81 ArActionLimiterTableSensor, 81 fire, 82 ArActionLimiterTableSensor, 81 fire, 82 ArActionLimiterTableSensor, 81 ArActionStallRecover, 83 ArActionStallRecover, 84 ArActionStall		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		
joystickInited, 70 setSpeeds, 70 setStopIfNoButtonPressed, 70 setThrottleParams, 70 ArActionJoydrive, 70 ArActionJoydrive, 71 fire, 72 getUseOSCal, 72 setUseOSCal, 72 ArActionKeydrive ArActionKeydrive, 74 activate, 74 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 getDesired, 74 getDesired, 83 ArActionStallRecover, 84 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 ArActionStallRecover, 85 ArActionStop, 85 setIncrements, 74 setRobot, 74 setSpeeds, 74 ArActionStop, 85 fire, 86 ArActionStop, 85 fire, 86 ArActionTurn		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	-	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		
setThrottleParams, 70 ArActionJoydrive, 70 ArActionJoydrive, 71 ArActionLimiterTableSensor, 81 fire, 72 getUseOSCal, 72 setUseOSCal, 72 ArActionStallRecover setUseOSCal, 72 ArActionKeydrive ArActionStallRecover, 83 ArActionKeydrive, 74 activate, 74 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 setSpeeds, 74 ArActionTurn 81 getDesired, 81 ArActionStallRecover ArActionStallRecover, 83 ArActionStallRecover, 84 fire, 84 ArActionStallRecover, 84 fire, 84 ArActionStop SetDesired, 74 ArActionStop SetDesired, 75 GetDesired, 85 ArActionStop, 85 ArActionSto	- · · · · · · · · · · · · · · · · · · ·	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		
ArActionJoydrive, 71 fire, 72 getUseOSCal, 72 setUseOSCal, 72 ArActionStallRecover setUseOSCal, 72 ArActionStallRecover, 83 ArActionKeydrive ArActionKeydrive, 74 activate, 74 ArActionStallRecover, 83 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 ArActionTurn ArActionLimiterTableSensor, 81 fire, 82 ArActionStallRecover, 83 ArActionStallRecover, 84 getDesired, 83 ArActionStallRecover, 84 fire, 84 ArActionStallRecover, 84 fire, 84 ArActionStop ArActionStop ArActionStop, 85 getDesired, 85 ArActionStop,		
$\begin{array}{llll} & \text{fire, } 72 \\ & \text{getUseOSCal, } 72 \\ & \text{setUseOSCal, } 72 \\ & \text{ArActionStallRecover} \\ & \text{setUseOSCal, } 72 \\ & & \sim & \text{ArActionStallRecover, } 83 \\ & \text{ArActionKeydrive} \\ & & \sim & \text{ArActionStallRecover, } 84 \\ & & \sim & \text{ArActionKeydrive, } 74 \\ & & \text{activate, } 74 \\ & & \text{ArActionStallRecover, } 83 \\ & \text{ArActionKeydrive, } 74 \\ & & \text{deactivate, } 74 \\ & & \text{deactivate, } 74 \\ & & \text{down, } 75 \\ & & \text{getDesired, } 74 \\ & & \text{giveUpKeys, } 75 \\ & & \text{getDesired, } 85 \\ & & \text{giveUpKeys, } 75 \\ & & \text{left, } 75 \\ & & \text{getDesired, } 85 \\ & & \text{right, } 75 \\ & & \text{setIncrements, } 74 \\ & & \text{setSpeeds, } 74 \\ & & \text{fire, } 86 \\ & & \text{ArActionTurn} \\ \end{array}$		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		
setUseOSCal, 72 ArActionStallRecover, 83 ArActionKeydrive ArActionStallRecover, 84 getDesired, 83 ArActionStallRecover, 83 ArActionStallRecover, 83 ArActionStallRecover, 83 ArActionStallRecover, 84 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 ArActionStallRecover, 84 ArActionStallRecover, 84 ArActionStallRecover, 84 ArActionStallRecover, 84 ArActionStallRecover, 85 ArActionStop, 85		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	-	
~ArActionKeydrive, 74 activate, 74 ArActionStallRecover, 83 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 getDesired, 83 ArActionStallRecover, 84 fire, 84 ArActionStop ~ArActionStop setDesired, 85 ArActionStop, 85 getDesired, 85 ArActionStop, 85 ArA		
activate, 74 ArActionStallRecover, 83 ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 ArActionStallRecover, 84 ArActionStop \sim ArActionStop \sim ArActionStop, 85 getDesired, 85 ArActionStop, 85 setIncrements, 74 ArActionStop, 85 setRobot, 74 ArActionStop, 85		
ArActionKeydrive, 74 deactivate, 74 down, 75 getDesired, 74 giveUpKeys, 75 left, 75 right, 75 setIncrements, 74 setSpeeds, 74 ArActionStallRecover, 84 fire, 84 ArActionStop \sim ArActionStop, 85 ArActionStop, 85 getDesired, 85 ArActionStop, 85 fire, 86 ArActionStop, 85	• • •	
$\begin{array}{lll} \text{deactivate, } 74 & \text{fire, } 84 \\ \text{down, } 75 & \text{ArActionStop} \\ \text{getDesired, } 74 & \sim & \text{ArActionStop, } 85 \\ \text{giveUpKeys, } 75 & \text{ArActionStop, } 85 \\ \text{left, } 75 & \text{getDesired, } 85 \\ \text{right, } 75 & \text{ArActionStop, } 85 \\ \text{setIncrements, } 74 & \text{ArActionStop, } 85 \\ \text{setRobot, } 74 & \text{fire, } 86 \\ \text{setSpeeds, } 74 & \text{ArActionTurn} \end{array}$		
$\begin{array}{llll} \operatorname{down}, 75 & \operatorname{ArActionStop} \\ \operatorname{getDesired}, 74 & \sim \operatorname{ArActionStop}, 85 \\ \operatorname{giveUpKeys}, 75 & \operatorname{ArActionStop}, 85 \\ \operatorname{left}, 75 & \operatorname{getDesired}, 85 \\ \operatorname{right}, 75 & \operatorname{ArActionStop}, 85 \\ \operatorname{setIncrements}, 74 & \operatorname{ArActionStop}, 85 \\ \operatorname{setRobot}, 74 & \operatorname{fire}, 86 \\ \operatorname{setSpeeds}, 74 & \operatorname{ArActionTurn} \end{array}$		
$\begin{array}{lll} {\rm getDesired, 74} & \sim & {\rm ArActionStop, 85} \\ {\rm giveUpKeys, 75} & {\rm ArActionStop, 85} \\ {\rm left, 75} & {\rm getDesired, 85} \\ {\rm right, 75} & {\rm ArActionStop, 85} \\ {\rm setIncrements, 74} & {\rm ArActionStop, 85} \\ {\rm setRobot, 74} & {\rm fire, 86} \\ {\rm setSpeeds, 74} & {\rm ArActionTurn} \end{array}$		
giveUpKeys, 75 left, 75 getDesired, 85 right, 75 setIncrements, 74 setRobot, 74 setSpeeds, 74 ArActionStop, 85 fire, 86 ArActionTurn		-
left, 75 getDesired, 85 right, 75 ArActionStop, 85 setIncrements, 74 ArActionStop, 85 setRobot, 74 fire, 86 setSpeeds, 74 ArActionTurn	,	- · · · · · · · · · · · · · · · · · · ·
right, 75 ArActionStop, 85 setIncrements, 74 ArActionStop, 85 setRobot, 74 fire, 86 setSpeeds, 74 ArActionTurn		- ·
setIncrements, 74 ArActionStop, 85 setRobot, 74 fire, 86 setSpeeds, 74 ArActionTurn		
setRobot, 74 fire, 86 setSpeeds, 74 ArActionTurn	9 .	- :
setSpeeds, 74 ArActionTurn		• /
± '		
space, 75 ~ArActionTurn, 87	- · · · · · · · · · · · · · · · · · · ·	
	space, 75	~ArActionTurn, 87

ArActionTurn, 87	ArAMPTU
getDesired, 87	\sim ArAMPTU, 96
ArActionTurn, 87	ArAMPTU, 98
fire, 88	canZoom, 97
ArACTS_1_2	getMaxNegPan, 97
\sim ArACTS_1_2, 89	getMaxNegTilt, 97
actsHandler, 90	getMaxPosPan, 97
ArACTS_1_2, 89	getMaxPosTilt, 97
BLOB_DATA_SIZE, 90	getPan, 97
DATA_HEADER, 90	getTilt, 97
getData, 90	init, 96
getRobot, 89	pan, 96
isConnected, 89	panRel, 96
MAX_BLOBS, 90	panSlew, 97
MAX_DATA, 90	panTilt, 96
NUM_CHANNELS, 90	panTiltRel, 96
setRobot, 89	panrintitel, 30 pause, 97
ArACTS_1_2, 89	pause, 97 purge, 97
ActsConstants, 90	requestStatus, 97
•	
closePort, 91	resume, 97
getBlob, 91 getNumBlobs, 91	tilt, 96
,	tiltRel, 96
invert, 91	tiltSlew, 97
openPort, 91	Aramptu 08
receiveBlobInfo, 92	ArAMPTU, 98
requestPacket, 92	ArAMPTUCommands
requestQuit, 92	ABSPAN, 99
Aractsblob	ABSTILT, 99
~ArACTSBlob, 94	CONT, 99
ArACTSBlob, 94	INIT, 100
getArea, 94	PANSLEW, 100
getBottom, 94	PANTILT, 99
getLeft, 94	PANTILTDCCW, 99
getRight, 94	PANTILTDCW, 99
getTop, 94	PANTILTUCCW, 99
getXCG, 94	PANTILTUCW, 99
getYCG, 94	PAUSE, 99
$\log, 95$	PURGE, 99
setArea, 94	RELPANCCW, 99
setBottom, 95	RELPANCW, 99
setLeft, 95	RELTILTD, 99
$\operatorname{setRight}, 95$	RELTILTU, 99
setTop, 95	RESP, 100
setXCG, 94	STATUS, 99
setYCG, 94	TILTSLEW, 100
ArACTSBlob, 94	ZOOM, 99

ArAMPTUCommands, 99	ArArgumentParser
ArAMPTUPacket	\sim ArArgumentParser, 107
\sim ArAMPTUPacket, 101	ArArgumentParser, 108
ArAMPTUPacket, 101	getArgc, 107
byte2ToBuf, 101	$\log, 107$
byteToBuf, 101	ArArgumentParser, 107
finalizePacket, 101	ArArgumentParser, 108
ArAMPTUPacket, 101	checkArgument, 108
getUnitNumber, 102	checkParameterArgument, 108
setUnitNumber, 102	ArASyncTask
ArArg	\sim ArASyncTask, 110
\sim ArArg, 103	ArASyncTask, 110
ArArg, 103	create, 110
BOOL, 105	run, 110
clearPointers, 104	runAsync, 110
DOUBLE, 105	stopRunning, 110
getBool, 104	ArASyncTask, 110
getDescription, 104	runInThisThread, 111
getDouble, 104	runThread, 111
getInt, 104	ArBasePacket
getName, 103	\sim ArBasePacket, 112
getPose, 104	ArBasePacket, 115
getString, 104	bufToByte, 113
INT, 105	bufToByte2, 113
INVALID, 105	bufToByte4, 113
log, 104	bufToUByte, 113
POSE, 105	bufToUByte2, 113
setBool, 104	bufToUByte4, 113
setDouble, 104	byte2ToBuf, 112
setInt, 104	byte4ToBuf, 112
setPose, 104	byteToBuf, 112
setString, 104	finalizePacket, 112
STRING, 105	getBuf, 114
ArArg, 103	getDataLength, 114
getType, 105	getDataReadLength, 114
Type, 105	getFooterLength, 114
ArArgumentBuilder	getHeaderLength, 114
~ArArgumentBuilder, 106	getLength, 114
add, 106	getMaxLength, 114
addPlain, 106	getReadLength, 114
ArArgumentBuilder, 106	log, 112
getArgc, 106	printHex, 112
getArgv, 106	setBuf, 114
log, 106	setHeaderLength, 114
removeArg, 106	setLength, 114
ArArgumentBuilder, 106	setReadLength, 114
O	G*,

uByte2ToBuf, 113	SAY, 120
uByte4ToBuf, 113	SETA, 119
uByteToBuf, 113	SETO, 119
ArBasePacket, 112	SETRA, 120
ArBasePacket, 115	SETRV, 120
bufToData, 116	SETSIMORIGINTH, 121
bufToStr, 116	SETSIMORIGINX, 121
dataToBuf, 116	SETSIMORIGINY, 121 SETSIMORIGINY, 121
duplicatePacket, 116	SETSIMORIGIN 1, 121 SETV, 119
empty, 116	•
resetRead, 117	SONAR, 120
strNToBuf, 117	SOUND, 121
strToBuf, 117	SOUNDTOG, 121
strToBufPadded, 117	STEP, 120
ArCommands	STOP, 120
	TCM2, 120
ADSEL, 120	TTY2, 120
BUMPSTALL, 120	VEL, 120
CALCOMP, 121	VEL2, 120
CLOSE, 119	ArCommands, 119
CONFIG, 120	Commands, 119
DCHEAD, 120	ArCondition
DHEAD, 120	\sim ArCondition, 122
DIGOUT, 120	ArCondition, 122
ENABLE, 119	broadcast, 122
ENCODER, 120	getError, 122
ENDSIM, 120	signal, 122
ESTOP, 120	STATUS_FAILED, 123
GETAUX, 120	
GRIPPER, 120	STATUS_FAILED
GRIPPERPACREQUEST,	DESTROY, 123
120	STATUS_FAILED_INIT, 123
GRIPPERVAL, 120	STATUS_MUTEX_FAILED,
HEAD, 120	123
IOREQUEST, 120	STATUS_MUTEX_FAILED
JOYDRIVE, 120	INIT, 123
LOADPARAM, 120	STATUS_WAIT_INTR, 123
LOADWORLD, 120	STATUS_WAIT_TIMEDOUT,
MOVE, 119	123
OPEN, 119	timedWait, 122
PLAYLIST, 121	wait, 122
POLLING, 119	ArCondition, 122
PTUPOS, 120	typedef, 123
PULSE, 119	ArDeviceConnection
RESETSIMTOORIGIN, 121	~ArDeviceConnection, 124
ROTATE, 120	ArDeviceConnection, 124
RVEL, 120	openSimple, 125
100 EL, 120	opensimple, 125

a-1a a- a a	
STATUS_CLOSED_ERROR,	immedExec, 131
125	indepMove, 133
STATUS_CLOSED	init, 130
NORMALLY, 125	initMon, 132
STATUS_NEVER_OPENED,	limitEnforce, 131
125	lowerPanSlew, 133
STATUS_OPEN, 125	lowerTiltSlew, 133
STATUS_OPEN_FAILED, 125	lowMotPower, 133
ArDeviceConnection, 124	lowStatPower, 133
close, 126	MAX_PAN_ACCEL, 135
getOpenMessage, 126	MAX_PAN_SLEW, 135
getStatus, 126	MAX_TILT, 135
getStatusMessage, 126	MAX_TILT_ACCEL, 135
getTimeRead, 127	MAX_TILT_SLEW, 135
isTimeStamping, 127	MIN_PAN, 135
read, 127	MIN_PAN_ACCEL, 135
Status, 125	MIN_PAN_SLEW, 135
write, 128	MIN_TILT, 135
write, 128 writePacket, 128	MIN_TILT_ACCEL, 135
ArDPPTU	MIN_TILT_ACCEL, 135 MIN_TILT_SLEW, 135
~ArdPPTU, 130	myPan, 134
ArDPPTU, 130	offStatPower, 132
awaitExec, 132	pan, 131
basePanSlew, 133	panAccel, 133
baseTiltSlew, 133	panRel, 131
canZoom, 130	panSlew, 134
disableReset, 130	panSlewRel, 134
disMon, 132	panTilt, 131
enMon, 132	panTiltRel, 131
factorySet, 131	regMotPower, 133
getBasePanSlew, 134	regStatPower, 132
getBaseTiltSlew, 134	resetAll, 131
getMaxNegPan, 132	resetCalib, 130
getMaxNegTilt, 132	resetPan, 131
getMaxPosPan, 132	resetTilt, 131
getMaxPosTilt, 132	restoreSet, 131
getPan, 134	saveSet, 131
getPanAccel, 134	slave $Exec, 132$
getPanSlew, 134	tilt, 131
getTilt, 134	tiltAccel, 133
getTiltAccel, 134	tiltRel, 131
getTiltSlew, 134	tiltSlew, 134
haltAll, 132	tiltSlewRel, 134
haltPan, 132	upperPanSlew, 133
haltTilt, 132	upperTiltSlew, 133
highMotPower, 133	velMove, 133
,	,

ArDPPTU, 130	ArFunctor1C, 145, 146
blank, 135	invoke, 146
ArDPPTUCommands	setP1, 146
ACCEL, 136	setThis, 146, 147
BASE, 136	ArFunctor2
CONTROL, 136	~ArFunctor2, 148
DELIM, 136	invoke, 148
DISABLE, 136	ArFunctor2, 148
ENABLE, 136	invoke, 149
FACTORY, 136	ArFunctor2C
HALT, 136	~ArFunctor2C, 150
IMMED, 137	ArFunctor2C, 150–152
INIT, 136	invoke, 150
LIMIT, 137	ArFunctor2C, 150
MONITOR, 137	ArFunctor2C, 151, 152
OFFSET, 137	invoke, 153
PAN, 137	setP1, 153
RESET, 137	setP2, 153
SPEED, 137	setThis, 154
TILT, 137	ArFunctor3
UPPER, 137	~ArFunctor3, 155
VELOCITY, 137	invoke, 155
ArDPPTUCommands, 136	ArFunctor3, 155
ArDPPTUPacket	invoke, 156
\sim ArDPPTUPacket, 138	ArFunctor3C
ArDPPTUPacket, 138	~ArFunctor3C, 158
finalizePacket, 138	ArFunctor3C, 157, 159, 160
ArDPPTUPacket, 138	invoke, 158
areMotorsEnabled	ArFunctor3C, 157
ArRobot, 359	ArFunctor3C, 159, 160
areSonarsEnabled	invoke, 161
ArRobot, 359	setP1, 162
ArFunctor	setP2, 162
\sim ArFunctor, 139	setP3, 162
invoke, 139	setThis, 162, 163
ArFunctor, 139	ArFunctorASyncTask
ArFunctor1	~ArFunctorASyncTask, 164
\sim ArFunctor1, 142	ArFunctorASyncTask, 164
invoke, 142	runThread, 164
ArFunctor1, 142	ArFunctorASyncTask, 164
invoke, 143	ArFunctorC
ArFunctor1C	\sim ArFunctorC, 165
~ArFunctor1C, 144	ArFunctorC, 165, 166
ArFunctor1C, 144–146	invoke, 165
invoke, 144	ArFunctorC, 165
ArFunctor1C, 144	ArFunctorC, 166

setThis, 166	ArGlobalRetFunctor1, 185
ArGlobalFunctor	invokeR, 185
~ArGlobalFunctor, 168	setP1, 186
ArGlobalFunctor, 168, 169	ArGlobalRetFunctor2
invoke, 168	~ArGlobalRetFunctor2, 187
ArGlobalFunctor, 168	ArGlobalRetFunctor2, 187,
ArGlobalFunctor, 169	188
ArGlobalFunctor1	invokeR, 187
~ArGlobalFunctor1, 170	ArGlobalRetFunctor2, 187
ArGlobalFunctor1, 170, 171	ArGlobalRetFunctor2, 188
invoke, 170	invokeR, 189
ArGlobalFunctor1, 170	setP1, 189
ArGlobalFunctor1, 171	setP2, 189
invoke, 171	ArGlobalRetFunctor3
setP1, 171	~ArGlobalRetFunctor3, 191
ArGlobalFunctor2	ArGlobalRetFunctor3, 191–
~ArGlobalFunctor2, 173	193
ArGlobalFunctor2, 173, 174	invokeR, 191
invoke, 173	ArGlobalRetFunctor3, 191
ArGlobalFunctor2, 173	ArGlobalRetFunctor3, 192,
ArGlobalFunctor2, 174	193
invoke, 175	invokeR, 193, 194
setP1, 175	setP1, 194
	set 1, 194 set P2, 194
setP2, 175	
ArGlobalFunctor3	setP3, 195
~ArGlobalFunctor3, 177	ArGripper
ArGlobalFunctor3, 177–179	~ArGripper, 196
invoke, 177	ArGripper, 198
ArGlobalFunctor3, 177	connectHandler, 198
ArGlobalFunctor3, 178, 179	GENIO, 198
invoke, 179, 180	GRIPPAC, 198
setP1, 180	logState, 198
setP2, 180	NOGRIPPER, 198
setP3, 181	packetHandler, 198
ArGlobalRetFunctor	QUERYTYPE, 198
~ArGlobalRetFunctor, 182	USERIO, 198
ArGlobalRetFunctor, 182, 183	ArGripper, 196
invokeR, 182	ArGripper, 198
ArGlobalRetFunctor, 182	getBreakBeamState, 199
ArGlobalRetFunctor, 183	getGraspTime, 199
ArGlobalRetFunctor1	getGripState, 199
~ArGlobalRetFunctor1, 184	getMSecSinceLastPacket, 199
ArGlobalRetFunctor1, 184,	getPaddleState, 199
185	getType, 200
invokeR, 184	gripClose, 200
ArGlobalRetFunctor1, 184	gripOpen, 200

gripperDeploy, 200 gripperHalt, 201 gripperStore, 201	SigHandleMethod, 206 signalHandlerCB, 206 uninit, 209
gripPressure, 200	ArInterpolation
gripStop, 200	~ArInterpolation, 210
isGripMoving, 201	addReading, 210
isLiftMaxed, 201	ArInterpolation, 210
isLiftMoving, 201	getNumberOfReadings, 210
liftCarry, 201	reset, 210
liftDown, 202	setNumberOfReadings, 210
liftStop, 202	ArInterpolation, 210
liftUp, 202	getPose, 211
setType, 202	ArIrrfDevice
Type, 198	\sim ArIrrfDevice, 212
ArGripperCommands	ArIrrfDevice, 212
GRIP_CLOSE, 203	setCumulativeMaxRange, 212
GRIP_OPEN, 203	setRobot, 212
GRIP_PRESSURE, 203	ArIrrfDevice, 212
GRIP_STOP, 203	packetHandler, 213
GRIPPER_DEPLOY, 203	ArJoyHandler
GRIPPER_HALT, 203	\sim ArJoyHandler, 214
GRIPPER_STORE, 203	ArJoyHandler, 216
LIFT_CARRY, 204	getSpeeds, 215
LIFT_DOWN, 203	getStats, 215
LIFT_STOP, 203	haveJoystick, 214
LIFT_UP, 203	haveZAxis, 214
ArGripperCommands, 203	init, 214
Commands, 203	setSpeeds, 214
Aria, 205	setStats, 215
addInitCallBack, 206	ArJoyHandler, 214
addRobot, 205	ArJoyHandler, 216
addUninitCallBack, 207	endCal, 216
delRobot, 205	getAdjusted, 216
exit, 207	getAxis, 217
findRobot, 207	getButton, 217
getDirectory, 207	getDoubles, 217
getKeyHandler, 206	getNumAxes, 217
getRobotList, 206	getNumButtons, 218
getRunning, 208	getUnfiltered, 218
init, 208	getUseOSCal, 218
setDirectory, 208	setUseOSCal, 218
setKeyHandler, 206	startCal, 219
shutdown, 209	ArKeyHandler
SIGHANDLE_NONE, 206	\sim ArKeyHandler, 220
SIGHANDLE_SINGLE, 206	ArKeyHandler, 221
SIGHANDLE_THREAD, 206	BACKSPACE, 221

checkKeys, 220	myY2, 227
DOWN, 221	newEndPoints, 226
ENTER, 221	ArLineSegment, 226
ESCAPE, 221	getPerpPoint, 227, 228
F1, 221	intersects, 228
F2, 221	ArListPos
F3, 221	FIRST, 230
F4, 221	LAST, 230
getKey, 220	ArListPos, 230
LEFT, 221	Pos, 230
restore, 220	ArLog
RIGHT, 221	close, 231
SPACE, 221	Colbert, 232
TAB, 221	File, 232
UP, 221	logPlain, 231
ArKeyHandler, 220	None, 232
addKeyHandler, 222	Normal, 231
ArKeyHandler, 221	StdErr, 232
KEY, 221	StdOut, 232
remKeyHandler, 222	Terse, 231
ArLine	Verbose, 232
\sim ArLine, 224	ArLog, 231
ArLine, 224	init, 232
getA, 224	$\log, 232$
getB, 224	LogLevel, 231
getC, 224	LogType, 232
makeLinePerp, 224	ArLogFileConnection
newParameters, 224	~ArLogFileConnection, 233
newParametersFrom-	ArLogFileConnection, 233
Endpoints, 224	internalOpen, 234
ArLine, 224	OPEN_FILE_NOT_FOUND,
intersects, 225	234
ArLineSegment	OPEN_NOT_A_LOG_FILE,
~ArLineSegment, 226	234
ArLineSegment, 226	openSimple, 233
getA, 226	ArLogFileConnection, 233
getB, 227	close, 234
getC, 227	getLogFile, 234
getX1, 226	getOpenMessage, 235
getX2, 226	getStatus, 235
getY1, 226	getTimeRead, 235
getY2, 226	isTimeStamping, 236
linePointIsInSegment, 227	Open, 234
myX1, 227	open, 236
myX2, 227	read, 236
myY1, 227	write, 237
, 11, 221	W1100, 201

ArMath	ArMode, 246
angleBetween, 238	baseActivate, 246
pointRotate, 238	baseDeactivate, 246
random, 239	help, 246
ArMath, 238	ArModeCamera
addAngle, 239	\sim ArModeCamera, 248
atan2, 239	activate, 248
$\cos, 240$	ArModeCamera, 248
degToRad, 240	deactivate, 248
distanceBetween, 240	userTask, 248
fabs, 240	ArModeCamera, 248
fixAngle, 241	help, 249
radToDeg, 241	ArModeGripper
roundInt, 241	\sim ArModeGripper, 250
$\sin, 241$	activate, 250
squaredDistanceBetween, 242	ArModeGripper, 250
subAngle, 242	deactivate, 250
ArmGood	userTask, 250
ArP2Arm, 276	ArModeGripper, 250
ArmHoming	help, 251
ArP2Arm, 276	ArModeSonar
ArmInited	\sim ArModeSonar, 252
ArP2Arm, 276	activate, 252
ArmJoint1	ArModeSonar, 252
ArP2Arm, 276	deactivate, 252
ArmJoint2	userTask, 252
ArP2Arm, 276	ArModeSonar, 252
ArmJoint3	help, 253
ArP2Arm, 276	ArModeTeleop
ArmJoint4	\sim ArModeTeleop, 254
ArP2Arm, 276	activate, 254
ArmJoint5	ArModeTeleop, 254
ArP2Arm, 276	deactivate, 254
ArmJoint6	ArModeTeleop, 254
ArP2Arm, 276	help, 254
ArMode	${\bf Ar Mode Unguar ded Teleop}$
\sim ArMode, 244	\sim ArModeUnguardedTeleop,
activate, 245	256
ArMode, 246	activate, 256
baseHelp, 245	ArModeUnguardedTeleop, 256
deactivate, 245	deactivate, 256
getKey, 245	ArModeUnguardedTeleop, 256
getKey2, 245	help, 256
getName, 244	ArModeWander
userTask, 245	\sim ArModeWander, 258
ArMode, 244	activate, 258

A M. 1 M. 1 M. 1 OF 0	A N + C 0.00
ArModeWander, 258	ArNetServer, 268
deactivate, 258	close, 268
ArModeWander, 258	internalEcho, 269
help, 258	internal Greeting, 268
ArModule	internalHelp, 269
\sim ArModule, 260	internal Quit, 269
ArModule, 260	internalShutdown, 269
exit, 260	isOpen, 268
getRobot, 260	runOnce, 268
myRobot, 260	sendToAllClientsPlain, 268
setRobot, 260	ArNetServer, 268
ArModule, 260	addCommand, 269
init, 261	open, 269
ArModuleLoader	remCommand, 270
closeAll, 263	sendToAllClients, 270
STATUS_ALREADY	ArNetServerConnection
LOADED, 264	~ArNetServerConnection, 271
STATUS_EXIT_FAILED, 264	ArNetServerConnection, 271
STATUS_FAILED_OPEN, 264	doEcho, 271
STATUS_INIT_FAILED, 264	getEcho, 271
STATUS_INVALID, 264	getSocket, 271
STATUS_NOT_FOUND, 264	setEcho, 271
STATUS_SUCCESS, 264	•
ArModuleLoader, 263	ArNetServerConnection, 271
	readString, 271
close, 264	<u> </u>
close, 264 load, 264	ArP2Arm
load, 264	ArP2Arm ~ArP2Arm, 273
load, 264 reload, 265	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED,
load, 264	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278
load, 264 reload, 265 Status, 264 ArmPower	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277
load, 264 reload, 265 Status, 264	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex \sim ArMutex, 266	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex \sim ArMutex, 266 ArMutex, 266	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED, 267	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED, 267 STATUS_FAILED_INIT, 267	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276 ArmJoint5, 276 ArmJoint6, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED_INIT, 267 unlock, 266	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276 ArmJoint6, 276 ArmPower, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED_INIT, 267 unlock, 266 ArMutex, 266 ArMutex, 266	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276 ArmJoint6, 276 ArmPower, 276 ArP2Arm, 273
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED_INIT, 267 unlock, 266 ArMutex, 266 ArMutex, 266 lock, 267	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276 ArmJoint6, 276 ArmPower, 276 ArP2Arm, 273 COMM_FAILED, 277
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED_INIT, 267 unlock, 266 ArMutex, 266 lock, 267 Status, 267	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276 ArmJoint6, 276 ArmPower, 276 ArP2Arm, 273 COMM_FAILED, 277 convertDegToTicks, 275
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED_INIT, 267 unlock, 266 ArMutex, 266 lock, 267 Status, 267 tryLock, 267	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmHoming, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276 ArmJoint6, 276 ArmPower, 276 Arp2Arm, 273 COMM_FAILED, 277 convertDegToTicks, 275 convertTicksToDeg, 276
load, 264 reload, 265 Status, 264 ArmPower ArP2Arm, 276 ArMutex ~ArMutex, 266 ArMutex, 266 getError, 266 getMutex, 266 STATUS_ALREADY LOCKED, 267 STATUS_FAILED_INIT, 267 unlock, 266 ArMutex, 266 lock, 267 Status, 267	ArP2Arm ~ArP2Arm, 273 ALREADY_CONNECTED, 278 ALREADY_INITED, 277 ArmGood, 276 ArmHoming, 276 ArmInited, 276 ArmJoint1, 276 ArmJoint2, 276 ArmJoint3, 276 ArmJoint4, 276 ArmJoint5, 276 ArmJoint6, 276 ArmPower, 276 ArP2Arm, 273 COMM_FAILED, 277 convertDegToTicks, 275

COULD_NOT_SET_UP	State, 277
PORT, 278	StatusType, 278
getArmVersion, 275	stop, 283
getJoint, 275	uninit, 284
getJointPos, 275	ArPose
getJointPosTicks, 275	\sim ArPose, 285
getLastStatusTime, 275	ArPose, 285, 286
getMoving, 275	getTh, 286
getRobot, 275	getThRad, 286
getStatus, 275	getX, 286
InfoPacket, 277	getY, 286
INVALID_JOINT, 278	$\log, 286$
INVALID_POSITION, 278	setTh, 285
isGood, 275	setThRad, 285
isPowered, 275	setX, 285
NO_ARM_FOUND, 277	setY, 285
NOT_CONNECTED, 278	ArPose, 285
NOT_INITED, 277	ArPose, 286
NumJoints, 276	findAngleTo, 287
park, 274	findDistanceTo, 287
ROBOT_NOT_SETUP, 277	getPose, 287
setPacketCB, 275	setPose, 288
setRobot, 273	squaredFindDistanceTo, 288
setStoppedCB, 275	ArPoseWithTime, 289
StatusContinuous, 278	ArPref
StatusOff, 278	\sim ArPref, 290
StatusPacket, 277	ArPref, 290
StatusSingle, 278	Boolean, 292
SUCCESS, 277	Double, 292
ArP2Arm, 273	Integer, 292
checkArm, 278	String, 292
home, 279	ArPref, 290
init, 279	getBool, 292
moveStep, 279	getBoolSet, 292
moveStepTicks, 279	getDouble, 292
moveTo, 280	getDoubleSet, 293
moveToTicks, 280	getInt, 293
moveVel, 281	getIntSet, 293
PacketType, 277	getSetCount, 293
powerOff, 281	getString, 294
powerOn, 282	getStringSet, 294
requestInfo, 282	setBool, 294
requestInit, 282	setBoolSet, 294
requestStatus, 283	setDouble, 295
setAutoParkTimer, 283	setDoubleSet, 295
setGripperParkTimer, 283	setInt, 295
scionppen arkinner, 200	SCUIII, 200

setIntSet, 296	ArRangeBuffer
setString, 296	\sim ArRangeBuffer, 305
ValType, 292	ArRangeBuffer, 307
ArPriorityResolver	clear, 306
\sim ArPriorityResolver, 297	clearOlderThan, 306
ArPriorityResolver, 297	clearOlderThanSeconds, 306
ArPriorityResolver, 297	getPoseTaken, 305
ArPTZ	getSize, 305
~ArPTZ, 298	reset, 306
ArPTZ, 301	setPoseTaken, 305
canGetRealPanTilt, 299	ArRangeBuffer, 305
	addReading, 307
canGetRealZoom, 299	٠,
canZoom, 298	applyTransform, 307
connectHandler, 300	ArRangeBuffer, 307
getAuxPort, 300	beginInvalidationSweep, 307
getDeviceConnection, 300	beginRedoBuffer, 307
getMaxNegPan, 299	endInvalidationSweep, 308
getMaxNegTilt, 299	endRedoBuffer, 308
getMaxPosPan, 299	getBuffer, 308
getMaxPosTilt, 299	getClosestBox, 309
getMaxZoom, 300	getClosestPolar, 309
getMinZoom, 300	invalidateReading, 310
getPan, 299	redoReading, 310
getRealPan, 299	setSize, 311
getRealTilt, 299	ArRangeDevice
getRealZoom, 299	~ArRangeDevice, 312
getTilt, 299	addReading, 312
getZoom, 299	ArRangeDevice, 315
init, 298	clearCumulativeOlderThan,
pan, 298	314
panRel, 298	clearCumulativeOlderThan-
panTilt, 298	Seconds, 314
panTiltRel, 298	clearCumulativeReadings, 314
÷ /	9 .
sensorInterpHandler, 300	clearCurrentReadings, 314
tilt, 298	getCumulativeBuffer, 313
tiltRel, 298	getCumulativeRangeBuffer,
zoom, 299	313
zoomRel, 299	getCurrentBuffer, 313
ArPTZ, 298	getCurrentRangeBuffer, 313
ArPTZ, 301	getMaxRange, 314
packetHandler, 302	getName, 312
readPacket, 302	getRobot, 312
robotPacketHandler, 302	setMaxRange, 314
sendPacket, 303	setRobot, 312
setAuxPort, 303	ArRangeDevice, 312
setDeviceConnection, 303	applyTransform, 315

ArRangeDevice, 315	ArRetFunctor1
cumulativeReadingBox, 315	\sim ArRetFunctor1, 328
cumulativeReadingPolar, 316	invokeR, 328
currentReadingBox, 316	ArRetFunctor1, 328
currentReadingPolar, 317	invokeR, 329
getRawReadings, 318	ArRetFunctor1C
lockDevice, 318	\sim ArRetFunctor1C, 330
setCumulativeBufferSize, 318	ArRetFunctor1C, 330–332
setCurrentBufferSize, 318	invokeR, 330
tryLockDevice, 319	ArRetFunctor1C, 330
unlockDevice, 319	ArRetFunctor1C, 331, 332
ArRangeDeviceThreaded	invokeR, 332
\sim ArRangeDeviceThreaded,	setP1, 332
320	setThis, 333
ArRangeDeviceThreaded, 320	ArRetFunctor2
getRunning, 320	\sim ArRetFunctor2, 334
getRunningWithLock, 320	invokeR, 334
run, 320	ArRetFunctor2, 334
runAsync, 320	invokeR, 335
runThread, 320	ArRetFunctor2C
stopRunning, 320	~ArRetFunctor2C, 337
ArRangeDeviceThreaded, 320	ArRetFunctor2C, 336, 338, 339
lockDevice, 321	invokeR, 337
tryLockDevice, 321	ArRetFunctor2C, 336
unlockDevice, 321	ArRetFunctor2C, 338, 339
ArRecurrentTask	invokeR, 339
\sim ArRecurrentTask, 323	setP1, 340
ArRecurrentTask, 323	setP2, 340
go, 323	setThis, 340
reset, 323	ArRetFunctor3
ArRecurrentTask, 323	\sim ArRetFunctor3, 342
done, 324	invokeR, 342
runThread, 324	ArRetFunctor3, 342
task, 324	invokeR, 343
ArResolver	ArRetFunctor3C
\sim ArResolver, 325	~ArRetFunctor3C, 346
ActionMap, 325	ArRetFunctor3C, 345, 347–349
getDescription, 325	invokeR, 346
getName, 325	ArRetFunctor3C, 345
resolve, 325	ArRetFunctor3C, 347–349
ArResolver, 325	invokeR, 349
ArRetFunctor	setP1, 350
\sim ArRetFunctor, 327	setP2, 350
invoke, 327	setP3, 350
invokeR, 327	setThis, 350, 351
ArRetFunctor, 327	ArRetFunctorC

\sim ArRetFunctorC, 352	getResolver, 366
ArRetFunctorC, 352, 353	getRightVel, 359
invokeR, 352	getRobotDiagonal, 359
ArRetFunctorC, 352	getRobotName, 358
ArRetFunctorC, 353	getRobotRadius, 359
setThis, 353	getRobotSubType, 358
ArRobot	<u> </u>
	getRobotType, 358
~ArRobot, 355	getRotVel, 359
addRangeDevice, 363	getRunExitListCopy, 370
areMotorsEnabled, 359	getSonarPacCount, 361
areSonarsEnabled, 359	getStallValue, 359
ArRobot, 371	getTh, 358
dropConnection, 369	getVel, 359
failedConnect, 369	getX, 358
getAnalog, 360	getY, 358
getAnalogPortSelected, 360	handlePacket, 370
getBatteryVoltage, 359	hasFrontBumpers, 361
getClosestSonarNumber, 362	hasRearBumpers, 361
getClosestSonarRange, 362	hasTableSensingIR, 360
getCompass, 360	incCounter, 368
getCounter, 368	isCycleChained, 367
getDigIn, 360	is Left Break Beam Triggered,
getDigOut, 360	361
getEncoderPose, 361	isLeftMotorStalled, 359
getFlags, 359	isLeftTableSensing-
getHeadingDoneDiff, 357	IRTriggered, 360
getIOAnalog, 360	is Right Break Beam Triggered,
getIOAnalogSize, 360	361
getIODigIn, 360	isRightMotorStalled, 359
getIODigInSize, 360	isRightTableSensing-
getIODigOut, 360	IRTriggered, 360
getIODigOutSize, 360	keyHandlerExit, 369
getIOPacketTime, 361	lock, 368
getKeyHandler, 368	logActions, 366
getLeftVel, 359	madeConnection, 369
getMaxRotVel, 358	processEncoderPacket, 369
getMaxTransVel, 358	processIOPacket, 369
getMotorPacCount, 361	processMotorPacket, 369
getMoveDoneDist, 357	processNewSonar, 369
getName, 362	processParamFile, 370
getNumFrontBumpers, 361	setCycleChained, 367
getNumRearBumpers, 361	setHeadingDoneDiff, 357
getNumSonar, 361	setMoveDoneDist, 357
getPose, 358	setName, 362
getPoseInterpNumReadings,	setName, 502 setPoseInterpNumReadings,
getPoseinterpnumReadings, 367	1
901	367

setResolver, 366 enableMotors, 382 setUpPacketHandlers, 369 findAction, 382 setUpSyncList, 369 findRangeDevice, 382, 383 tryLock, 368 findTask, 383 findUserTask, 383, 384 unlock, 368 WAIT_CONNECTED, 370 getActionMap, 384 WAIT_FAIL, 370 getConnectionCycleMultiplier, WAIT_FAILED_CONN, 370 WAIT_INTR, 370 getConnectionTimeoutTime, WAIT_RUN_EXIT, 370 384 WAIT_TIMEDOUT, 370 getControl, 385 ArRobot, 355 getCycleTime, 385 actionHandler, 371 getDeviceConnection, 385 addAction, 371 getDirectMotionPrecedenceaddConnectCB, 372 Time, 385 addDisconnectNormallyCB, getEncoderCorrectionCallback, 386 addDisconnectOnErrorCB, getEncoderTransform, 386 372 getLastPacketTime, 386 addFailedConnectCB, 373 getPoseInterpPosition, 386 addPacketHandler, 373 getRangeDeviceList, 387 addRunExitCB, 374 getRobotParams, 387 addSensorInterpTask, 374 getSonarRange, 387 addUserTask, 375 getSonarReading, 387 applyTransform, 375 getStateReflectionRefresh-Time, 388 ArRobot, 371 asyncConnect, 375 getSyncTaskRoot, 388 asyncConnectHandler, 376 getToGlobalTransform, 389 attachKeyHandler, 377 getToLocalTransform, 389 blockingConnect, 377 hasRangeDevice, 389 init, 389 checkRangeDevicesCumulativeBox, 378 isConnected, 389 checkRangeDevicesCumulaisDirectMotion, 389 tivePolar, 378 isHeadingDone, 390 checkRangeDevicesCurrentisMoveDone, 390 Box, 379 isRunning, 390 checkRangeDevicesCurrentPoisSonarNew, 391 lar, 379 loadParamFile, 391 clearDirectMotion, 380 logAllTasks, 391 com, 380 logUserTasks, 391 com2Bytes, 380 loopOnce, 391 comInt, 381 move, 392 comStr, 381 moveTo, 392 comStrN, 381 packetHandler, 392 disableMotors, 381 remAction, 393 disconnect, 382 remConnectCB, 393

D: AM II CD	1 4110 177 177 171
remDisconnectNormallyCB,	wakeAllConnWaitingThreads,
393	404
remDisconnectOnErrorCB,	wakeAllRunExitWaitingTh-
394	reads, 405
remFailedConnectCB, 394	wakeAllWaitingThreads, 405
remPacketHandler, 394	ArRobotPacket
remRangeDevice, 394, 395	~ArRobotPacket, 406
remRunExitCB, 395	ArRobotPacket, 407
remSensorInterpTask, 395	calcCheckSum, 406
remUserTask, 395, 396	finalizePacket, 406
robotLocker, 396	getID, 406
robotUnlocker, 396	getTimeReceived, 406
run, 396	setID, 406
runAsync, 396	setTimeReceived, 406
setConnectionCycleMultiplier,	verifyCheckSum, 406
397	ArRobotPacket, 406
setConnectionTimeoutTime,	ArRobotPacket, 407
397	ArRobotPacketReceiver
setCycleTime, 397	\sim ArRobotPacketReceiver, 408
setDeadReconPose, 398	ArRobotPacketReceiver, 409
setDeltaHeading, 398	getDeviceConnection, 408
setDeviceConnection, 398	isAllocatingPackets, 408
setDirectMotionPrecedence-	setDeviceConnection, 408
Time, 398	ArRobotPacketReceiver, 408
setEncoderCorrectionCall-	ArRobotPacketReceiver, 409
back, 399	receivePacket, 409
setEncoderTransform, 399	ArRobotPacketSender
	\sim ArRobotPacketSender, 411
setHeading, 400	ArRobotPacketSender, 412
setMaxRotVel, 400	getDeviceConnection, 411
setMaxTransVel, 400	setDeviceConnection, 411
setRotVel, 401	ArRobotPacketSender, 411
setStateReflectionRefresh-	ArRobotPacketSender, 412
Time, 401	com, 412
setVel, 401	com2Bytes, 412
setVel2, 402	comInt, 413
stateReflector, 402	comStr, 413
stop, 402	comStrN, 413
stopRunning, 403	ArRobotParams
waitForConnect, 403	\sim ArRobotParams, 415
waitForConnectOrConnFail,	ArRobotParams, 415
403	getAngleConvFactor, 416
waitForRunExit, 404	getClassName, 415
WaitState, 370	getDiffConvFactor, 416
wakeAllConnOrFailWait-	getDistConvFactor, 416
ingThreads, 404	getLaserFlipped, 417

getLaserPort, 417	getSensorX, 419
getLaserPossessed, 417	getSensorY, 419
getLaserPowerControlled, 417	getThTaken, 420
getLaserX, 417	getX, 419
getLaserY, 417	getXTaken, 420
getMaxRotVelocity, 415	getY, 419
getMaxVelocity, 415	getYTaken, 420
getNumSonar, 416	ArSensorReading, 419
getRangeConvFactor, 416	applyTransform, 421
getRequestIOPackets, 415	ArSensorReading, 421
getRobotDiagonal, 415	getCounterTaken, 421
getRobotRadius, 415	getRange, 421
getSonarTh, 417	getSensorPosition, 422
getSonarX, 417	isNew, 422
getSonarY, 417	newData, 422
getSubClassName, 415	resetSensorPosition, 422
getVel2Divisor, 416	ArSensorReading::newData
getVelConvFactor, 416	ArSensorReading, 420
hasMoveCommand, 415	ArSerialConnection
haveFrontBumpers, 416	~ArSerialConnection, 424
haveNewTableSensingIR, 416	ArSerialConnection, 424
haveRearBumpers, 416	getCTS, 425
haveSonar, 417	getDCD, 425
haveTableSensingIR, 416	OPEN_ALREADY_OPEN,
init, 415	426
isHolonomic, 415	OPEN_COULD_NOT_OPEN
numFrontBumpers, 416	PORT, 426
numRearBumpers, 416	OPEN_COULD_NOT_SET
ArRobotParams, 415	BAUD, 426
ArSectors	OPEN_COULD_NOT_SET
~ArSectors, 418	UP_PORT, 426
ArSectors, 418	OPEN_INVALID_BAUD
clear, 418	RATE, 426
didAll, 418	openSimple, 424
update, 418	ArSerialConnection, 424
ArSectors, 418	close, 426
ArSensorReading	getBaud, 426
ArSensorReading, 421	getHardwareControl, 426
ArSensorReading::newData,	getOpenMessage, 426
420	getPort, 427
getEncoderPoseTaken, 419	getStatus, 427
getPose, 419	getTimeRead, 427
getPoseTaken, 419	isTimeStamping, 428
getSensorDX, 420	Open, 426
getSensorDY, 420	open, 428
getSensorTh, 419	read, 428
000000000000000000000000000000000000000	1000, 1=0

setBaud, 429	INCREMENT_ONE, 437
setHardwareControl, 429	isConnected, 432
setPort, 429	isControllingPower, 434
write, 429	isLaserFlipped, 434
ArSick	isUsingSim, 434
~ArSick, 431	madeConnection, 435
ArSick, 431	processPacket, 435
ArSick::getFilterClean-	robotConnectCallback, 435
CumulativeInterval,	runOnce, 435
433	runThread, 435
ArSick::getFilterCumulative-	sensorInterpCallback, 435
CleanDistance, 433	setDeviceConnection, 432
ArSick::getFilterCumulative-	setMinRange, 432
MaxDistance, 433	setRobot, 435
ArSick::getFilterCumulative-	setSensorPosition, 431, 432
NearDistance, 433	simPacketHandler, 435
ArSick::setFilterClean-	STATE_CHANGE_BAUD, 437
CumulativeInterval,	STATE_CHANGE_BAUD, 437 STATE_CONFIGURE, 437
433	STATE-CONFIGURE, 437 STATE-CONNECTED, 437
ArSick::setFilterCumulative-	STATE LOWNE CIED, 437 STATE LINIT, 437
CleanDist, 433	STATE_INIT, 437 STATE_INSTALL_MODE, 437
ArSick::setFilterCumulative-	STATE_NONE, 437
MaxDist, 433 ArSick::setFilterCumulative-	STATE_SET_MODE, 437 STATE_START_READINGS,
	•
NearDist, 433	437
BAUD19200, 436	STATE_WAIT_FOR
BAUD38400, 436	CONFIGURE_ACK,
BAUD9600, 436	437
DEGREES100, 437	STATE_WAIT_FOR
DEGREES180, 437	INSTALL_MODE_ACK,
dropConnection, 435	437
failedConnect, 435	STATE_WAIT_FOR
filterAddAndClean-	POWER_ON, 437
Cumulative, 436	STATE_WAIT_FOR_SET
filterFarCumulative, 436	MODE_ACK, 437
getDegrees, 434	STATE_WAIT_FOR_START
getDeviceConnection, 432	ACK, 437
getIncrement, 435	switchState, 436
getLastReadingTime, 434	tryingToConnect, 432
getMinRange, 432	ArSick, 431
getSensorPosition, 432	addConnectCB, 437
getSensorPositionTh, 432	add Disconnect Normally CB,
getSensorPositionX, 432	438
getSensorPositionY, 432	addDisconnectOnErrorCB,
getSickPacCount, 433	438
INCREMENT_HALF, 437	addFailedConnectCB, 439

asyncConnect, 439	getDegDiff, 444
BaudRate, 436	getDistDiff, 444
blockingConnect, 440	robotTask, 444
configure, 440	setDegDiff, 444
configureShort, 440	setDistDiff, 444
Degrees, 436	takeReading, 444
disconnect, 440	ArSickLogger, 444
filterReadings, 441	addTagToLog, 445
getConnectionTimeoutTime,	ArSickLogger, 445
441	ArSickPacket
getFilterNearDist, 441	\sim ArSickPacket, 447
Increment, 437	ArSickPacket, 447
internalConnectHandler, 441	calcCRC, 447
internalConnectSim, 442	finalizePacket, 447
remConnectCB, 442	getID, 447
remDisconnectNormallyCB,	getTimeReceived, 448
442	setTimeReceived, 448
remDisconnectOnErrorCB,	verifyCRC, 447
442	ArSickPacket, 447
remFailedConnectCB, 442	duplicatePacket, 448
runOnRobot, 443	getReceivedAddress, 448
setConnectionTimeoutTime,	getSendingAddress, 449
443	resetRead, 449
setFilterNearDist, 443	setSendingAddress, 449
State, 437	ArSickPacketReceiver
ArSick::getFilterCleanCumulativeInterval	~ArSickPacketReceiver, 450
ArSick, 433	ArSickPacketReceiver, 451
ArSick::getFilterCumulativeCleanDistance	getDeviceConnection, 450
ArSick, 433	isAllocatingPackets, 450
ArSick::getFilterCumulativeMaxDistance	setDeviceConnection, 450
ArSick, 433	ArSickPacketReceiver, 450
ArSick::getFilterCumulativeNearDistance	ArSickPacketReceiver, 451
ArSick, 433	receivePacket, 451
ArSick::setFilterCleanCumulativeInterval	ArSignalHandler
ArSick, 433	~ArSignalHandler, 453
ArSick::setFilterCumulativeCleanDist	blockAllThisThread, 454
ArSick, 433	nameSignal, 454
ArSick::setFilterCumulativeMaxDist	ArSignalHandler, 453
ArSick, 433	addHandlerCB, 455
ArSick::setFilterCumulativeNearDist	block, 455
ArSick, 433	blockCommon, 455
ArSickLogger	blockCommonThisThread, 456
~ArSickLogger, 444	createHandlerNonThreaded,
addGoal, 444	456
addTagToLogPlain, 444	createHandlerThreaded, 456
ArSickLogger, 445	delHandlerCB, 456
MIDICALOSSEI, 440	delifandierOD, 450

.TT 11 488	N DI 1 400
getHandler, 457	setNonBlock, 463
handle, 457	setReuseAddress, 463
runThread, 457	sockAddrIn, 462
unblock, 457	sockAddrLen, 464
unblockAll, 458	writeStringPlain, 463
unhandle, 458	ArSocket, 461
ArSimpleConnector	ArSocket, 465
~ArSimpleConnector, 459	copy, 465
ArSimpleConnector, 459	init, 465
connectRobot, 459	read, 465
logOptions, 459	readString, 466
parseArgs, 459	shutdown, 466
ArSimpleConnector, 459	transfer, 466
setupLaser, 460	write, 467
setupRobot, 460	writeString, 467
ArSocket	ArSonarDevice
\sim ArSocket, 461	\sim ArSonarDevice, 468
accept, 462	ArSonarDevice, 468
addrHost, 464	processReadings, 468
ArSocket, 461, 465	setCumulativeMaxRange, 468
close, 462	setRobot, 468
connect, 461	ArSonarDevice, 468
connectTo, 461, 462	addReading, 469
copy, 461	ArSonyPacket
create, 461	ArSonyPacket, 470
findValidPort, 461	byte2ToBuf, 470
getError, 463	uByteToBuf, 470
getErrorStr, 463	ArSonyPacket, 470
getFD, 463	byte2ToBufAtPos, 470
getHostName, 464	ArSonyPTZ
getSockName, 462	canZoom, 472
getType, 463	getMaxNegPan, 473
hostAddr, 463	getMaxNegTilt, 473
hostToNetOrder, 464	getMaxPosPan, 473
inAddr, 462	getMaxPosTilt, 473
inPort, 462	getMaxZoom, 473
inToA, 464	getMinZoom, 473
maxHostNameLen, 464	getPan, 473
netToHostOrder, 464	getTilt, 473
open, 461	getZoom, 473
ourInitialized, 464	init, 472
recvFrom, 462	MAX_PAN, 474
sendTo, 462	MAX_TILT, 474
setBroadcast, 462	MAX_TILT, 474 MAX_ZOOM, 474
setDoClose, 463	MAX-200M, 474 MIN-ZOOM, 474
setLinger, 462	
Settinger, 402	pan, 472

panRel, 472	getHost, 483
panTilt, 472	getOpenMessage, 483
panTiltRel, 472	getPort, 483
tilt, 472	getStatus, 484
tilt, 472 tiltRel, 472	getTimeRead, 484
zoom, 473	isTimeStamping, 484
zoomRel, 473	Open, 482
ArSonyPTZ, 472	open, 484
ArSyncTask	read, 485
ArSyncTask, 477	setSocket, 485
getFunctor, 476	write, 485
getName, 476	ArThread
getState, 475	~ArThread, 487
setState, 475	ArThread, 487
ArSyncTask, 475	cancel, 488
\sim ArSyncTask, 477	cancelAll, 489
addNewBranch, 477	create, 487
addNewLeaf, 477	detach, 488
ArSyncTask, 477	getBlockAllSignals, 488
find, 478	getFunc, 488
findNonRecursive, 478, 479	getJoinable, 488
$\log, 479$	getRunning, 488
run, 479	getRunningWithLock, 488
ArTaskState	getThread, 488
ACTIVE, 480	join, 488
FAILURE, 480	joinAll, 489
INIT, 480	lock, 488
RESUME, 480	myRunning, 489
SUCCESS, 480	setRunning, 488
SUSPEND, 480	STATUS_ALREADY
USER_START, 480	DETATCHED, 490
ArTaskState, 480	STATUS_FAILED, 490
State, 480	STATUS_INVALID, 490
ArTcpConnection	STATUS_JOIN_SELF, 490
~ArTcpConnection, 481	STATUS_NO_SUCH
ArTcpConnection, 481	THREAD, 490
getSocket, 482	STATUS_NORESOURCE, 490
internalOpen, 482	stopAll, 489
OPEN_BAD_HOST, 482	stopRunning, 487
OPEN_CON_REFUSED, 483	tryLock, 488
OPEN_NET_FAIL, 482	unlock, 488
OPEN_NO_ROUTE, 483	yieldProcessor, 489
openSimple, 481	ArThread, 487
setStatus, 482	init, 490
ArTcpConnection, 481	self, 490
close, 483	Status, 490
01050, 400	Duanus, 400

ArTime	BIT4, 499
\sim ArTime, 491	BIT5, 499
addMSec, 492	BIT6, 499
ArTime, 491	BIT7, 499
getMSec, 492	BIT8, 499
getSec, 492	BIT9, 499
isAfter, 491	REGKEY_CLASSES_ROOT,
isAt, 491	499
isBefore, 491	REGKEY_CURRENT
$\log, 492$	CONFIG, 499
mSecSince, 491	REGKEY_CURRENT_USER,
mSecTo, 491	499
secSince, 491	REGKEY_LOCAL
secTo, 491	MACHINE, 499
setMSec, 492	REGKEY_USERS, 499
setSec, 492	ArUtil, 497
setToNow, 491	appendSlash, 500
ArTime, 491	BITS, 499
ArTransform	deleteSet, 500
~ArTransform, 493	deleteSetPairs, 500
ArTransform, 493	escapeSpaces, 500
doTransform, 493	findFile, 500
getTh, 494	fixSlashes, 501
ArTransform, 493	fixSlashesBackward, 501
doInvTransform, 494	fixSlashesForward, 501
doTransform, 494, 495	getStringFromFile, 501
setTransform, 495	getStringFromRegistry, 501
ArTypes	getTime, 502
Byte, 496	REGKEY, 499
Byte2, 496	sizeFile, 502
Byte4, 496	sleep, 502
UByte, 496	splitString, 503
UByte2, 496	strcmp, 503, 504
UByte4, 496	stripDir, 505
	<u> </u>
ArTypes, 496 ArUtil	stripFile, 505 ArVCC4
BIT0, 499	~ArVCC4, 506
BIT1, 499	addErrorCB, 507
BIT10, 499	ArVCC4, 510
BIT11, 499	CAM_ERROR_BUSY, 509
BIT12, 499	CAM_ERROR_MODE, 509
BIT13, 499	CAM_ERROR_NONE, 509
BIT14, 499	CAM_ERROR_PARAM, 509
BIT15, 499	CAM_ERROR_UNKNOWN,
BIT2, 499	509
BIT3, 499	canZoom, 507

connectHandler, 506 getMaxNegPan, 507 petMaxNegTilt, 507 petMaxPanSlew, 508 getMaxPosPan, 507 petMaxPosPan, 507 petMaxPosTilt, 507 petMaxPosTilt, 507 petMaxPosTilt, 507 petMaxTiltSlew, 508 petMaxTiltSlew, 508 petMaxTiltSlew, 508 petMaxTiltSlew, 508 petMaxTiltSlew, 513 petMaxTiltSlew, 508 petMaxTiltSlew, 513 petMaxTiltSlew, 508
getMaxNegTilt, 507 PANTILT, 513 getMaxPanSlew, 508 PANTILTREQ, 513 getMaxPosPan, 507 POWER, 513 getMaxPosTilt, 507 RESPONSE, 513
getMaxPanSlew, 508 PANTILTREQ, 513 getMaxPosPan, 507 POWER, 513 getMaxPosTilt, 507 RESPONSE, 513
getMaxPosPan, 507 POWER, 513 getMaxPosTilt, 507 RESPONSE, 513
getMaxPosTilt, 507 RESPONSE, 513
,
getMaxZoom, 509 SLEWREQ, 513
getMinPanSlew, 508 STOP, 513
getMinTiltSlew, 508 TILTSLEW, 513
getMinZoom, 509 ZOOM, 513
getPan, 508 ZOOMSTOP, 513
getPanSlew, 508 ArVCC4Commands, 512
getRealPanTilt, 507 Command, 513
getRealZoomPos, 507 Command, 513
getTilt, 508 AIVCC4Packet, 514
getTiltSlew, 508 ArVCC4Packet, 514
getZoom, 508 ATVCC41 acket, 514 getZoom, 508 finalizePacket, 514
haltPanTilt, 508 ArVCC4Packet, 514
haltZoom, 508 AIVCC4Facket, 514 asyncConnect
init, 506 asyncconnect ArRobot, 375
,
- · · · · · · · · · · · · · · · · · · ·
preparePacket, 508 ArRobot, 377 remErrorCB, 507 awaitExec
and the second of the second o
tilt, 506 ArDPPTU, 132 tiltRel, 507
tiltSlew, 508 BACKSPACE
wasError, 509 ArKeyHandler, 221
zoom, 507 BASE
ArVCC4, 506 ArDPPTUCommands, 136
ArVCC4, 510 baseActivate
Error, 509 ArMode, 246
packetHandler, 510 baseDeactivate
readPacket, 511 ArMode, 246
ArVCC4Commands baseHelp
ANGLEREQ, 513 ArMode, 245
CONTROL, 513 basePanSlew
DELIM, 513 ArDPPTU, 133
DEVICEID, 513 baseTiltSlew
FOOTER, 513 ArDPPTU, 133
HEADER, 513 BAUD19200

ArSick, 436	ArDPPTU, 135
BAUD38400	BLOB_DATA_SIZE
ArSick, 436	ArACTS_1_2, 90
BAUD9600	block
ArSick, 436	ArSignalHandler, 455
BaudRate	blockAllThisThread
ArSick, 436	ArSignalHandler, 454
beginInvalidationSweep	blockCommon
ArRangeBuffer, 307	ArSignalHandler, 455
beginRedoBuffer	${\bf block Common This Thread}$
ArRangeBuffer, 307	ArSignalHandler, 456
BIT0	blockingConnect
ArUtil, 499	ArRobot, 377
BIT1	ArSick, 440
ArUtil, 499	BOOL
BIT10	ArArg, 105
ArUtil, 499	Boolean
BIT11	ArPref, 292
ArUtil, 499	broadcast
BIT12	ArCondition, 122
ArUtil, 499	bufToByte
BIT13	ArBasePacket, 113
ArUtil, 499	bufToByte2
BIT14	ArBasePacket, 113
ArUtil, 499	bufToByte4
BIT15	ArBasePacket, 113
ArUtil, 499	bufToData
BIT2	ArBasePacket, 116
ArUtil, 499 BIT3	bufToStr
	ArBasePacket, 116 bufToUByte
ArUtil, 499 BIT4	ArBasePacket, 113
ArUtil, 499	bufToUByte2
BIT5	ArBasePacket, 113
ArUtil, 499	bufToUByte4
BIT6	ArBasePacket, 113
ArUtil, 499	BUMPSTALL
BIT7	ArCommands, 120
ArUtil, 499	Byte
BIT8	ArTypes, 496
ArUtil, 499	Byte2
BIT9	ArTypes, 496
ArUtil, 499	byte2ToBuf
BITS	ArAMPTUPacket, 101
ArUtil, 499	ArBasePacket, 112
blank	ArSonyPacket, 470

byte2ToBufAtPos	checkKeys
ArSonyPacket, 470	ArKeyHandler, 220
Byte4	checkParameterArgument
ArTypes, 496	ArArgumentParser, 108
byte4ToBuf	checkRangeDevicesCumulativeBox
ArBasePacket, 112	ArRobot, 378
byteToBuf	check Range Devices Cumulative Polar
ArAMPTUPacket, 101	ArRobot, 378
ArBasePacket, 112	checkRangeDevicesCurrentBox
	ArRobot, 379
calcCheckSum	checkRangeDevicesCurrentPolar
ArRobotPacket, 406	ArRobot, 379
calcCRC	clear
ArSickPacket, 447	ArRangeBuffer, 306
CALCOMP	ArSectors, 418
ArCommands, 121	clearCumulativeOlderThan
CAM_ERROR_BUSY	ArRangeDevice, 314
ArVCC4, 509	clearCumulativeOlderThanSeconds
CAM_ERROR_MODE	ArRangeDevice, 314
ArVCC4, 509	clearCumulativeReadings
CAM_ERROR_NONE	ArRangeDevice, 314
ArVCC4, 509	clearCurrentReadings
CAM_ERROR_PARAM	ArRangeDevice, 314
ArVCC4, 509	clearDirectMotion
CAM_ERROR_UNKNOWN	ArRobot, 380
ArVCC4, 509	clearOlderThan
cancel	ArRangeBuffer, 306
ArThread, 488	clearOlderThanSeconds
cancelAll	ArRangeBuffer, 306
ArThread, 489	clearPointers
cancelGoal	ArArg, 104
ArActionGoto, 58	CLOSE
can Get Real Pan Tilt	ArCommands, 119
ArPTZ, 299	close
canGetRealZoom	ArDeviceConnection, 126
ArPTZ, 299	ArLog, 231
canZoom	ArLogFileConnection, 234
ArAMPTU, 97	ArModuleLoader, 264
ArDPPTU, 130	ArNetServer, 268
ArPTZ, 298	ArSerialConnection, 426
ArSonyPTZ, 472	ArSocket, 462
ArVCC4, 507	ArTcpConnection, 483
checkArgument	closeAll
ArArgumentParser, 108	ArModuleLoader, 263
checkArm	closePort
ArP2Arm, 278	ArACTS_1_2, 91

Colbert	ArP2Arm, 275
ArLog, 232	convert Ticks To Deg
com	ArP2Arm, 276
ArRobot, 380	copy
ArRobotPacketSender, 412	ArSocket, 461, 465
com2Bytes	cos
ArRobot, 380	ArMath, 240
ArRobotPacketSender, 412	COULD_NOT_OPEN_PORT
comInt	ArP2Arm, 277
ArRobot, 381	COULD_NOT_SET_UP_PORT
ArRobotPacketSender, 413	ArP2Arm, 278
COMM_FAILED	create
ArP2Arm, 277	ArASyncTask, 110
Command	ArSocket, 461
ArVCC4Commands, 513	ArThread, 487
Commands	create Handler Non Threaded
ArCommands, 119	ArSignalHandler, 456
ArGripperCommands, 203	createHandlerThreaded
comStr	ArSignalHandler, 456
ArRobot, 381	$\operatorname{cumulativeReadingBox}$
ArRobotPacketSender, 413	ArRangeDevice, 315
comStrN	cumulativeReadingPolar
ArRobot, 381	ArRangeDevice, 316
ArRobotPacketSender, 413	currentReadingBox
CONFIG	ArRangeDevice, 316
ArCommands, 120	currentReadingPolar
configure	ArRangeDevice, 317
ArSick, 440	DATA HEADED
configureShort	DATA_HEADER
ArSick, 440	ArACTS_1_2, 90 dataToBuf
Angeritat 461	ArBasePacket, 116
ArSocket, 461 connectHandler	DCHEAD
ArGripper, 198	ArCommands, 120
ArPTZ, 300	deactivate
ArVCC4, 506	ArAction, 41
connectRobot	ArActionGroup, 60
ArSimpleConnector, 459	ArActionKeydrive, 74
connectTo	ArMode, 245
ArSocket, 461, 462	ArModeCamera, 248
CONT	ArModeGripper, 250
ArAMPTUCommands, 99	ArModeSonar, 252
CONTROL	ArModeTeleop, 254
ArDPPTUCommands, 136	ArModeUnguardedTeleop, 256
ArVCC4Commands, 513	ArModeWander, 258
convertDegToTicks	Degrees

ArSick, 436	ArMath, 240
DEGREES100	doEcho
ArSick, 437	ArNetServerConnection, 271
DEGREES180	doInvTransform
ArSick, 437	ArTransform, 494
degToRad	done
ArMath, 240	ArRecurrentTask, 324
deleteSet	doTransform
ArUtil, 500	ArTransform, 493–495
deleteSetPairs	DOUBLE
ArUtil, 500	ArArg, 105
delHandlerCB	Double
ArSignalHandler, 456	ArPref, 292
DELIM	DOWN
ArDPPTUCommands, 136	ArKeyHandler, 221
ArVCC4Commands, 513	down
$\operatorname{delRobot}$	ArActionKeydrive, 75
Aria, 205	dropConnection
deltaHeading	ArRobot, 369
ArActionInput, 68	ArSick, 435
${\it delta} {\it Heading} {\it From} {\it Current}$	duplicatePacket
ArActionInput, 68	ArBasePacket, 116
deltaVel	ArSickPacket, 448
ArActionInput, 68	
detach	empty
ArThread, 488	ArBasePacket, 116
DEVICEID	ENABLE
ArVCC4Commands, 513	ArCommands, 119
DHEAD	ArDPPTUCommands, 136
ArCommands, 120	enableMotors
didAll	ArRobot, 382
ArSectors, 418	ENCODER
DIGOUT	ArCommands, 120
ArCommands, 120	$\operatorname{endAverage}$
DISABLE	ArActionDesired, 54
ArDPPTUCommands, 136	endCal
disableMotors	ArJoyHandler, 216
ArRobot, 381	endInvalidationSweep
disableReset	ArRangeBuffer, 308
ArDPPTU, 130	endRedoBuffer
disconnect	ArRangeBuffer, 308
ArRobot, 382	ENDSIM
ArSick, 440	ArCommands, 120
disMon	enMon
ArDPPTU, 132	Ardpptu, 132
distanceBetween	ENTER

ArKeyHandler, 221	ArVCC4Packet, 514
Error	find
ArVCC4, 509	ArSyncTask, 478
ESCAPE	findAction
ArKeyHandler, 221	ArRobot, 382
escapeSpaces	findAngleTo
ArUtil, 500	ArPose, 287
ESTOP	findDistanceTo
ArCommands, 120	ArPose, 287
exit	$\operatorname{findFile}$
Aria, 207	ArUtil, 500
ArModule, 260	findNonRecursive
D4	ArSyncTask, 478, 479
F1	$\operatorname{findRangeDevice}$
ArKeyHandler, 221	ArRobot, 382, 383
F2	$\operatorname{findRobot}$
ArKeyHandler, 221	Aria, 207
F3	$\operatorname{findTask}$
ArKeyHandler, 221	ArRobot, 383
F4	$\operatorname{findUserTask}$
ArKeyHandler, 221	ArRobot, 383, 384
fabs	$\operatorname{findValidPort}$
ArMath, 240	ArSocket, 461
FACTORY	fire
ArDPPTUCommands, 136	ArAction, 42
factorySet	ArActionAvoidFront, 44
ArdPPTU, 131	ArActionAvoidSide, 46
failedConnect	ArActionBumpers, 48
ArRobot, 369	ArActionConstantVelocity, 50
ArSick, 435	ArActionGoto, 59
FAILURE	ArActionInput, 69
ArTaskState, 480	ArActionJoydrive, 72
File	ArActionKeydrive, 75
ArLog, 232	ArActionLimiterBackwards,
filterAddAndCleanCumulative	78
ArSick, 436	ArActionLimiterForwards, 80
filterFarCumulative	ArActionLimiterTableSensor,
ArSick, 436	82
filterReadings	ArActionStallRecover, 84
ArSick, 441	ArActionStop, 86
finalizePacket	ArActionTurn, 88
ArAMPTUPacket, 101	FIRST
ArBasePacket, 112	ArListPos, 230
ArDPPTUPacket, 138	fixAngle
ArRobotPacket, 406	ArMath, 241
ArSickPacket, 447	fixSlashes

ArUtil, 501 ArDPPTU, 134 fixSlashesBackward getBaseTiltSlew ArUtil, 501 ArDPPTU, 134 fixSlashesForward getBatteryVoltage ArUtil, 501 ArRobot, 359 FOOTER getBaud ArVCC4Commands, 513 ArSerialConnection, 426 getBlob**GENIO** ArACTS_1_2, 91 ArGripper, 198 getBlockAllSignals getA ArThread, 488 ArLine, 224 getBool ArLineSegment, 226 ArArg, 104 getActionList ArPref, 292 ArActionGroup, 60 getBoolSet getActionMap ArPref, 292 ArRobot, 384 getBottom getAdjusted ArACTSBlob, 94 ArJoyHandler, 216 getBreakBeamState getAnalog ArGripper, 199 ArRobot, 360 getBuf getAnalogPortSelected ArBasePacket, 114 ArRobot, 360 getBuffer getAngleConvFactor ArRangeBuffer, 308 ArRobotParams, 416 getButton getArea ArJoyHandler, 217 ${\rm get} C$ ArACTSBlob, 94 getArg ArLine, 224 ArAction, 41 ArLineSegment, 227 getArgc getClassName ArArgumentBuilder, 106 ArRobotParams, 415 ArArgumentParser, 107 getCloseDist ArActionGoto, 58 getArgv ArArgumentBuilder, 106 getClosestBox getArmVersion ArRangeBuffer, 309 ArP2Arm, 275 getClosestPolar**GETAUX** ArRangeBuffer, 309 ArCommands, 120 getClosestSonarNumber getAuxPort ArRobot, 362 ArPTZ, 300 getClosestSonarRange getAxis ArRobot, 362 ArJoyHandler, 217 getCompass getB ArRobot, 360 getConnectionCycleMultiplier ArLine, 224 ArLineSegment, 227 ArRobot, 384 getBasePanSlew getConnectionTimeoutTime

ArRobot, 384	ArActionBumpers, 47
ArSick, 441	ArActionConstantVelocity, 49
getControl	ArActionGoto, 59
ArRobot, 385	ArActionInput, 68
getCounter	ArActionJoydrive, 71
ArRobot, 368	ArActionKeydrive, 74
getCounterTaken	ArActionLimiterBackwards,
ArSensorReading, 421	77
	ArActionLimiterForwards, 79
getCTS ArSonialConnection 425	ArActionLimiterTolwards, 79 ArActionLimiterTableSensor,
ArSerialConnection, 425 getCumulativeBuffer	81
0	_
ArRangeDevice, 313	ArActionStallRecover, 83
getCumulativeRangeBuffer	ArActionStop, 85
ArRangeDevice, 313	ArActionTurn, 87
getCurrentBuffer	getDeviceConnection
ArRangeDevice, 313	ArPTZ, 300
getCurrentRangeBuffer	ArRobot, 385
ArRangeDevice, 313	ArRobotPacketReceiver, 408
getCycleTime	ArRobotPacketSender, 411
ArRobot, 385	ArSick, 432
getData	ArSickPacketReceiver, 450
ArACTS_1_2, 90	getDiffConvFactor
getDataLength	ArRobotParams, 416
ArBasePacket, 114	$\operatorname{getDigIn}$
getDataReadLength	ArRobot, 360
ArBasePacket, 114	$\operatorname{getDigOut}$
getDCD	ArRobot, 360
ArSerialConnection, 425	${\tt getDirectMotionPrecedenceTime}$
getDegDiff	ArRobot, 385
ArSickLogger, 444	getDirectory
getDegrees	Aria, 207
ArSick, 434	getDistConvFactor
getDeltaHeading	ArRobotParams, 416
ArActionDesired, 52	getDistDiff
getDeltaHeadingDesiredChannel	ArSickLogger, 444
ArActionDesired, 53	getDouble
getDeltaHeadingStrength	ArArg, 104
ArActionDesired, 52	ArPref, 292
getDescription	getDoubles
ArAction, 41	ArJoyHandler, 217
ArArg, 104	getDoubleSet
ArResolver, 325	ArPref, 293
getDesired	getEcho
ArAction, 41	ArNetServerConnection, 271
ArActionAvoidFront, 43	getEncoderCorrectionCallback
ArActionAvoidSide, 45	ArRobot, 386
ATACHOHAVOIUBIUE, 49	AII(0)(0), 500

getEncoderPose	getID
ArRobot, 361	ArRobotPacket, 406
getEncoderPoseTaken	ArSickPacket, 447
ArSensorReading, 419	getIncrement
getEncoderTransform	ArSick, 435
ArRobot, 386	getInt
getError	ArArg, 104
ArCondition, 122	ArPref, 293
ArMutex, 266	getIntSet
ArSocket, 463	ArPref, 293
getErrorStr	getIOAnalog
ArSocket, 463	ArRobot, 360
getFD	getIOAnalogSize
ArSocket, 463	ArRobot, 360
getFilterNearDist	$\operatorname{getIODigIn}$
ArSick, 441	ArRobot, 360
$\operatorname{getFlags}$	${\it getIODigInSize}$
ArRobot, 359	ArRobot, 360
getFooterLength	$\operatorname{getIODigOut}$
ArBasePacket, 114	ArRobot, 360
getFunc	getIODigOutSize
ArThread, 488	ArRobot, 360
getFunctor	getIOPacketTime
ArSyncTask, 476	ArRobot, 361
$\operatorname{get} \operatorname{Goal}$	$\operatorname{getJoinable}$
ArActionGoto, 58	ArThread, 488
getGraspTime	$\operatorname{getJoint}$
ArGripper, 199	ArP2Arm, 275
getGripState	getJointPos
ArGripper, 199	ArP2Arm, 275
getHandler	getJointPosTicks
ArSignalHandler, 457	ArP2Arm, 275
getHardwareControl	getJoyHandler
ArSerialConnection, 426	ÅrActionJoydrive, 71
getHeaderLength	getKey
ArBasePacket, 114	ArKeyHandler, 220
getHeading	ArMode, 245
ArActionDesired, 52	getKey2
getHeadingDoneDiff	ArMode, 245
ArRobot, 357	getKeyHandler
getHeadingStrength	Aria, 206
ArActionDesired, 52	ArRobot, 368
•	,
getHost	getLaserFlipped
ArTcpConnection, 483	ArRobotParams, 417
getHostName	getLaserPort
ArSocket, 464	ArRobotParams, 417

	A DDDELL 100
getLaserPossessed	ArDPPTU, 132
ArRobotParams, 417	ArPTZ, 299
getLaserPowerControlled	ArSonyPTZ, 473
ArRobotParams, 417	ArVCC4, 507
getLaserX	getMaxPosTilt
ArRobotParams, 417	ArAMPTU, 97
getLaserY	ArDPPTU, 132
ArRobotParams, 417	ArPTZ, 299
getLastPacketTime	ArSonyPTZ, 473
ArRobot, 386	ArVCC4, 507
getLastReadingTime	getMaxRange
ArSick, 434	ArRangeDevice, 314
getLastStatusTime	getMaxRotVel
ArP2Arm, 275	ArActionDesired, 52
getLeft	ArRobot, 358
ArACTSBlob, 94	getMaxRotVelDesiredChannel
getLeftVel	ArActionDesired, 53
ArRobot, 359	getMaxRotVelocity
getLength	ArRobotParams, 415
ArBasePacket, 114	getMaxRotVelStrength
getLogFile	ArActionDesired, 52
ArLogFileConnection, 234	getMaxTiltSlew
getMaxLength	ArVCC4, 508
ArBasePacket, 114	getMaxTransVel
getMaxNegPan	ArRobot, 358
ArAMPTU, 97	getMaxVel
ArDPPTU, 132	ArActionDesired, 52
ArPTZ, 299	getMaxVelDesiredChannel
ArSonyPTZ, 473	ArActionDesired, 53
ArVCC4, 507	getMaxVelocity
getMaxNegTilt	ArRobotParams, 415
ArAMPTU, 97	getMaxVelStrength
ArDPPTU, 132	ArActionDesired, 52
ArPTZ, 299	getMaxZoom
ArSonyPTZ, 473	ArPTZ, 300
ArVCC4, 507	ArSonyPTZ, 473
getMaxNegVel	ArVCC4, 509 getMinPanSlew
ArActionDesired, 52 getMaxNegVelDesiredChannel	ArVCC4, 508
	*
ArActionDesired, 53	getMinRange
getMaxNegVelStrength	ArSick, 432
ArActionDesired, 52	getMinTiltSlew
getMaxPanSlew	ArVCC4, 508
ArVCC4, 508	getMinZoom
getMaxPosPan	ArPTZ, 300
ArAMPTU, 97	ArSonyPTZ, 473

ArVCC4, 509	ArAMPTU, 97
getMotorPacCount	ArdPPTU, 134
ArRobot, 361	ArPTZ, 299
getMoveDoneDist	ArSonyPTZ, 473
ArRobot, 357	ArVCC4, 508
getMoving	getPanAccel
ArP2Arm, 275	ArDPPTU, 134
- -	getPanSlew
getMSec	ArDPPTU, 134
ArTime, 492	
getMSecSinceLastPacket	ArVCC4, 508
ArGripper, 199	getPerpPoint
getMutex	ArLineSegment, 227, 228
ArMutex, 266	getPort
getName	ArSerialConnection, 427
ArAction, 41	ArTcpConnection, 483
ArArg, 103	getPose
ArMode, 244	ArArg, 104
ArRangeDevice, 312	ArInterpolation, 211
ArResolver, 325	ArPose, 287
ArRobot, 362	ArRobot, 358
ArSyncTask, 476	ArSensorReading, 419
$\operatorname{getNumArgs}$	${\tt getPoseInterpNumReadings}$
ArAction, 41	ArRobot, 367
$\operatorname{getNumAxes}$	getPoseInterpPosition
ArJoyHandler, 217	ArRobot, 386
getNumberOfReadings	getPoseTaken
ArInterpolation, 210	ArRangeBuffer, 305
getNumBlobs	ArSensorReading, 419
ArACTS_1_2, 91	getRange
getNumButtons	ArSensorReading, 421
ArJoyHandler, 218	getRangeConvFactor
$\operatorname{getNumFrontBumpers}$	ArRobotParams, 416
ArRobot, 361	getRangeDeviceList
${\it getNumRearBumpers}$	ArRobot, 387
ArRobot, 361	getRawReadings
getNumSonar	ArRangeDevice, 318
ArRobot, 361	getReadLength
ArRobotParams, 416	ArBasePacket, 114
getOpenMessage	getRealPan
ArDeviceConnection, 126	ArPTZ, 299
ArLogFileConnection, 235	${ m getRealPanTilt}$
ArSerialConnection, 426	ArVCC4, 507
ArTcpConnection, 483	getRealTilt
getPaddleState	ArPTZ, 299
ArGripper, 199	$\operatorname{getRealZoom}$
getPan	ArPTZ, 299
$\stackrel{\smile}{\sim}$,

getRealZoomPos	getSendingAddress
ArVCC4, 507	ArSickPacket, 449
getReceivedAddress	getSensorDX
ArSickPacket, 448	ArSensorReading, 420
getRequestIOPackets	getSensorDY
ArRobotParams, 415	ArSensorReading, 420
getResolver	getSensorPosition
ArRobot, 366	ArSensorReading, 422
getRight	ArSick, 432
ArACTSBlob, 94	getSensorPositionTh
getRightVel	ArSick, 432
ArRobot, 359	getSensorPositionX
getRobot	ArSick, 432
ArACTS_1_2, 89	getSensorPositionY
ArModule, 260	ArSick, 432
ArP2Arm, 275	
	getSensorTh ArSensorPending 410
ArRangeDevice, 312 getRobotDiagonal	ArSensorReading, 419 getSensorX
ArRobot, 359	~
,	ArSensorReading, 419
ArRobotParams, 415	getSensorY
getRobotList	ArSensorReading, 419
Aria, 206	getSetCount
getRobotName	ArPref, 293
ArRobot, 358	getSickPacCount
getRobotParams	ArSick, 433
ArRobot, 387	getSize
getRobotRadius	ArRangeBuffer, 305
ArRobot, 359	getSocket
ArRobotParams, 415	ArNetServerConnection, 271
getRobotSubType	ArTcpConnection, 482
ArRobot, 358	getSockName
getRobotType	ArSocket, 462
ArRobot, 358	getSonarPacCount
getRotVel	ArRobot, 361
ArRobot, 359	getSonarRange
getRunExitListCopy	ArRobot, 387
ArRobot, 370	getSonarReading
getRunning	ArRobot, 387
Aria, 208	getSonarTh
ArRangeDeviceThreaded, 320	ArRobotParams, 417
ArThread, 488	getSonarX
getRunningWithLock	ArRobotParams, 417
ArRangeDeviceThreaded, 320	getSonarY
ArThread, 488	ArRobotParams, 417
getSec	getSpeed
ArTime, 492	ArActionGoto, 58

$\operatorname{getSpeeds}$	ArPTZ, 299
ArJoyHandler, 215	ArSonyPTZ, 473
getStallValue	ArVCC4, 508
ArRobot, 359	getTiltAccel
getState	ArDPPTU, 134
ArSyncTask, 475	getTiltSlew
getStateReflectionRefreshTime	ArDPPTU, 134
ArRobot, 388	ArVCC4, 508
getStats	getTime
ArJoyHandler, 215	ArUtil, 502
getStatus	getTimeRead
ArDeviceConnection, 126	ArDeviceConnection, 127
ArLogFileConnection, 235	ArLogFileConnection, 235
ArP2Arm, 275	ArLogr neconnection, 233 ArSerialConnection, 427
ArTen Connection, 427	ArTcpConnection, 484
ArTcpConnection, 484	getTimeReceived
getStatusMessage	ArRobotPacket, 406
ArDeviceConnection, 126	ArSickPacket, 448
getStopIfNoButtonPressed	getToGlobalTransform
ArActionJoydrive, 70	ArRobot, 389
getString	getToLocalTransform
ArArg, 104	ArRobot, 389
ArPref, 294	getTop
getStringFromFile	ArACTSBlob, 94
ArUtil, 501	$\operatorname{getType}$
getStringFromRegistry	ArArg, 105
ArUtil, 501	ArGripper, 200
getStringSet	ArSocket, 463
ArPref, 294	getUnfiltered
getSubClassName	ArJoyHandler, 218
ArRobotParams, 415	$\operatorname{getUnitNumber}$
getSyncTaskRoot	ArAMPTUPacket, 102
ArRobot, 388	${\it getUseOSCal}$
$\operatorname{get} \operatorname{Th}$	ArActionJoydrive, 72
ArPose, 286	ArJoyHandler, 218
ArRobot, 358	getVel
ArTransform, 494	ArActionDesired, 51
getThRad	ArRobot, 359
ArPose, 286	getVel2Divisor
getThread	ArRobotParams, 416
ArThread, 488	getVelConvFactor
getThTaken	ArRobotParams, 416
ArSensorReading, 420	getVelDesiredChannel
getTilt	ArActionDesired, 53
ArAMPTU, 97	getVelStrength
ArDPPTU, 134	ArActionDesired, 52
, :	

get X	GRIPPER
ArPose, 286	ArCommands, 120
ArRobot, 358	GRIPPER_DEPLOY
ArSensorReading, 419	ArGripperCommands, 203
getX1	GRIPPER_HALT
ArLineSegment, 226	ArGripperCommands, 203
get X2	GRIPPER_STORE
ArLineSegment, 226	ArGripperCommands, 203
getXCG	gripperDeploy
ArACTSBlob, 94	ArGripper, 200
getXTaken	gripperHalt
ArSensorReading, 420	ArGripper, 201
getY	GRIPPERPACREQUEST
ArPose, 286	ArCommands, 120
ArRobot, 358	gripperStore
ArSensorReading, 419	ArGripper, 201
getY1	GRIPPERVAL
ArLineSegment, 226	ArCommands, 120
getY2	gripPressure
ArLineSegment, 226	ArGripper, 200
getYCG	gripStop
ArACTSBlob, 94	ArGripper, 200
getYTaken	ArGripper, 200
ArSensorReading, 420	HALT
getZoom	ArDPPTUCommands, 136
ArPTZ, 299	haltAll
ArSonyPTZ, 473	ArDPPTU, 132
ArVCC4, 508	haltPan
	ArDPPTU, 132
giveUpKeys	haltPanTilt
ArActionKeydrive, 75	
go	ArVCC4, 508
ArRecurrentTask, 323	haltTilt
GRIP_CLOSE	ArDPPTU, 132
ArGripperCommands, 203	haltZoom
GRIP_OPEN	ArVCC4, 508
ArGripperCommands, 203	handle
GRIP_PRESSURE	ArSignalHandler, 457
ArGripperCommands, 203	handlePacket
GRIP_STOP	ArRobot, 370
ArGripperCommands, 203	hasFrontBumpers
gripClose	ArRobot, 361
ArGripper, 200	hasMoveCommand
gripOpen	ArRobotParams, 415
ArGripper, 200	hasRangeDevice
GRIPPAC	ArRobot, 389
ArGripper, 198	hasRearBumpers

A D 1 + 961	
ArRobot, 361	incCounter
hasTableSensingIR	ArRobot, 368
ArRobot, 360	Increment
haveAchievedGoal	ArSick, 437
ArActionGoto, 58	INCREMENT_HALF
haveFrontBumpers	ArSick, 437
ArRobotParams, 416	INCREMENT_ONE
haveJoystick	ArSick, 437
ArJoyHandler, 214	indepMove
have New Table Sensing IR	ArDPPTU, 133
ArRobotParams, 416	InfoPacket
haveRearBumpers	ArP2Arm, 277
ArRobotParams, 416	INIT
haveSonar	ArAMPTUCommands, 100
ArRobotParams, 417	ArDPPTUCommands, 136
haveTableSensingIR	ArTaskState, 480
ArRobotParams, 416	ArVCC4Commands, 513
haveZAxis	init
ArJoyHandler, 214	ArAMPTU, 96
HEAD	ArDPPTU, 130
ArCommands, 120	Aria, 208
HEADER	ArJoyHandler, 214
ArVCC4Commands, 513	ArLog, 232
help	ArModule, 261
ArMode, 246	ArP2Arm, 279
ArModeCamera, 249	ArPTZ, 298
ArModeGripper, 251	ArRobot, 389
ArModeSonar, 253	ArRobotParams, 415
ArModeTeleop, 254	ArSocket, 465
ArModeUnguardedTeleop, 256	ArSonyPTZ, 472
ArModeWander, 258	ArThread, 490
highMotPower	ArVCC4, 506
ArDPPTU, 133	initMon
home	ArDPPTU, 132
ArP2Arm, 279	inPort
$\mathrm{host}\mathrm{Addr}$	ArSocket, 462
ArSocket, 463	INT
hostToNetOrder	ArArg, 105
ArSocket, 464	Integer
1112001100, 101	ArPref, 292
IMMED	internalConnectHandler
ArDPPTUCommands, 137	ArSick, 441
immedExec	internalConnectSim
ArDPPTU, 131	ArSick, 442
inAddr	internalEcho
ArSocket, 462	ArNetServer, 269
	1111.00.001701, 200

internalGreeting	ArGlobalRetFunctor2, 187,
ArNetServer, 268	189
internalHelp	ArGlobalRetFunctor3, 191,
ArNetServer, 269	193, 194
internalOpen	ArRetFunctor, 327
ArLogFileConnection, 234	ArRetFunctor1, 328, 329
ArTcpConnection, 482	ArRetFunctor1C, 330, 332
internalQuit	ArRetFunctor2, 334, 335
ArNetServer, 269	ArRetFunctor2C, 337, 339
internalShutdown	ArRetFunctor3, 342, 343
ArNetServer, 269	ArRetFunctor3C, 346, 349
intersects	ArRetFunctorC, 352
ArLine, 225	IOREQUEST
ArLineSegment, 228	ArCommands, 120
inToA	isActive
ArSocket, 464	ArAction, 40
INVALID	isAfter
ArArg, 105	ArTime, 491
INVALID_JOINT	isAllocatingPackets
ArP2Arm, 278	ArRobotPacketReceiver, 408
INVALID_POSITION	ArSickPacketReceiver, 450
ArP2Arm, 278	isAt
invalidateReading	ArTime, 491
ArRangeBuffer, 310	isBefore
invert	ArTime, 491
ArACTS_1_2, 91	isConnected
invoke	ArACTS_1_2, 89
ArFunctor, 139	ArRobot, 389
ArFunctor1, 142, 143	ArSick, 432
ArFunctor1C, 144, 146	isControllingPower
ArFunctor 1C, 144, 140 ArFunctor 2, 148, 149	ArSick, 434
ArFunctor2C, 150, 153	isCycleChained
ArFunctor3, 155, 156	ArRobot, 367
ArFunctor3C, 158, 161	isDirectMotion
	ArRobot, 389
ArFunctorC, 165	isGood
ArGlobalFunctor, 168	ArP2Arm, 275
ArGlobalFunctor1, 170, 171	isGripMoving
ArGlobalFunctor2, 173, 175	ArGripper, 201
ArGlobalFunctor3, 177, 179,	isHeadingDone
180	ArRobot, 390
ArRetFunctor, 327	isHolonomic
invokeR	ArRobotParams, 415
ArGlobalRetFunctor, 182	isInitted
ArGlobalRetFunctor1, 184,	ArVCC4, 506
185	isLaserFlipped

ArSick, 434	KEY
isLeftBreakBeamTriggered	ArKeyHandler, 221
ArRobot, 361	keyHandlerExit
isLeftMotorStalled	ArRobot, 369
	Alltobot, 309
ArRobot, 359	LAST
isLeftTableSensingIRTriggered	ArListPos, 230
ArRobot, 360	LEFT
isLiftMaxed	ArKeyHandler, 221
ArGripper, 201	left.
isLiftMoving	ArActionKeydrive, 75
ArGripper, 201	LIFT_CARRY
isMoveDone	ArGripperCommands, 204
ArRobot, 390	LIFT_DOWN
isNew	ArGripperCommands, 203
ArSensorReading, 422	LIFT_STOP
isOpen	ArGripperCommands, 203
ArNetServer, 268	LIFT_UP
isPowered	ArGripperCommands, 203
ArP2Arm, 275	liftCarry
isRightBreakBeamTriggered	ArGripper, 201
ArRobot, 361	liftDown
isRightMotorStalled	ArGripper, 202
ArRobot, 359	liftStop
isRightTableSensingIRTriggered	ArGripper, 202
ArRobot, 360	liftUp
isRunning	ArGripper, 202
ArRobot, 390	LIMIT
isSonarNew	ArDPPTUCommands, 137
ArRobot, 391	limitEnforce
isTimeStamping	ArDPPTU, 131
ArDeviceConnection, 127	linePointIsInSegment
ArLogFileConnection, 236	ArLineSegment, 227
ArSerialConnection, 428	load
ArTcpConnection, 484	ArModuleLoader, 264
isUsingSim	LOADPARAM
ArSick, 434	ArCommands, 120
	loadParamFile
join	ArRobot, 391
ArThread, 488	LOADWORLD
joinAll	ArCommands, 120
ArThread, 489	lock
JOYDRIVE	ArMutex, 267
ArCommands, 120	ArRobot, 368
joystickInited	ArThread, 488
ArActionJoydrive, 70	lockDevice

	A.D D 210	A A C/T/C 1 2 00
	ArRangeDevice, 318	ArACTS_1_2, 90 MAX_DATA
1	ArRangeDeviceThreaded, 321	
\log	AnAstion 41	ArACTS_1_2, 90 MAX_PAN
	ArAction, 41	
	ArACTSBlob, 95	ArSonyPTZ, 474 MAX_PAN_ACCEL
	ArArg, 104	
	ArArgumentBuilder, 106	ArDPPTU, 135
	ArArgumentParser, 107	MAX_PAN_SLEW
	ArBasePacket, 112	Ardpptu, 135
	ArLog, 232	MAX_STRENGTH
	ArPose, 286	ArActionDesiredChannel, 57
	ArSyncTask, 479	MAX_TILT
	ArTime, 492	Arden Der 135
logA	Actions	ArSonyPTZ, 474
	ArRobot, 366	MAX_TILT_ACCEL
$\log A$	AllTasks	Arden Grand
	ArRobot, 391	MAX_TILT_SLEW
Log.	Level	ArdPPTU, 135
_	ArLog, 231	MAX_ZOOM
logC	Options	ArSonyPTZ, 474
_	ArSimpleConnector, 459	maxHostNameLen
logF	Plain	ArSocket, 464
	ArLog, 231	merge
logS		ArActionDesired, 54
	ArGripper, 198	MIN_PAN
Log'	Type	ArDPPTU, 135
	ArLog, 232	MIN_PAN_ACCEL
logU	JserTasks	ArDPPTU, 135
	ArRobot, 391	MIN_PAN_SLEW
loop	Once	ArDPPTU, 135
	ArRobot, 391	MIN_STRENGTH
lowe	erPanSlew	ArActionDesiredChannel, 57
	ArDPPTU, 133	MIN_TILT
lowe	erTiltSlew	ArDPPTU, 135
	ArDPPTU, 133	MIN_TILT_ACCEL
lowl	MotPower	ArDPPTU, 135
	ArDPPTU, 133	MIN_TILT_SLEW
lows	StatPower	ArDPPTU, 135
	ArDPPTU, 133	MIN_ZOOM
		ArSonyPTZ, 474
mad	leConnection	MONITOR
	ArRobot, 369	ArDPPTUCommands, 137
	ArSick, 435	MOVE
mak	teLinePerp	ArCommands, 119
	ArLine, 224	move
MA	X_BLOBS	ArRobot, 392

moveStep	NOGRIPPER
ArP2Arm, 279	ArGripper, 198
moveStepTicks	None
ArP2Arm, 279	ArLog, 232
moveTo	Normal
ArP2Arm, 280	ArLog, 231
ArRobot, 392	NOT_CONNECTED
moveToTicks	ArP2Arm, 278
ArP2Arm, 280	NOT_INITED
moveVel	ArP2Arm, 277
ArP2Arm, 281	NUM_CHANNELS
mSecSince	ArACTS_1_2, 90
ArTime, 491	numFrontBumpers
mSecTo	ArRobotParams, 416
ArTime, 491	NumJoints
myPan	ArP2Arm, 276
ArDPPTU, 134	numRearBumpers
myRobot	ArRobotParams, 416
ArModule, 260	mitoboti arams, 410
myRunning	OFFSET
ArThread, 489	ArDPPTUCommands, 137
myX1	offStatPower
ArLineSegment, 227	ArDPPTU, 132
myX2	OPEN
ArLineSegment, 227	ArCommands, 119
myY1	Open
ArLineSegment, 227	ArLogFileConnection, 234
myY2	ArSerialConnection, 426
ArLineSegment, 227	ArTcpConnection, 482
TILEMOS ogmono,	open
nameSignal	ArLogFileConnection, 236
ArSignalHandler, 454	ArNetServer, 269
netToHostOrder	ArSerialConnection, 428
ArSocket, 464	ArSocket, 461
newData	ArTcpConnection, 484
ArSensorReading, 422	OPEN_ALREADY_OPEN
newEndPoints	ArSerialConnection, 426
ArLineSegment, 226	OPEN_BAD_HOST
newParameters	ArTcpConnection, 482
ArLine, 224	OPEN_CON_REFUSED
newParametersFromEndpoints	ArTcpConnection, 483
ArLine, 224	OPEN_COULD_NOT_OPEN
NO_ARM_FOUND	PORT
ArP2Arm, 277	ArSerialConnection, 426
NO_STRENGTH	OPEN_COULD_NOT_SET_BAUD
ArActionDesiredChannel, 57	ArSerialConnection, 426

OPEN_COULD_NOT_SET_UP	ArSonyPTZ, 472
PORT	ArVCC4, 506
ArSerialConnection, 426	PANSLEW
OPEN_FILE_NOT_FOUND	ArAMPTUCommands, 100
ArLogFileConnection, 234	ArVCC4Commands, 513
OPEN_INVALID_BAUD_RATE	panSlew
ArSerialConnection, 426	ArAMPTU, 97
OPEN_NET_FAIL	ArDPPTU, 134
ArTcpConnection, 482	ArVCC4, 508
OPEN_NO_ROUTE	panSlewRel
ArTcpConnection, 483	ArDPPTU, 134
OPEN_NOT_A_LOG_FILE	PANTILT
ArLogFileConnection, 234	ArAMPTUCommands, 99
openPort	ArVCC4Commands, 513
ArACTS_1_2, 91	panTilt
openSimple	ArAMPTU, 96
ArDeviceConnection, 125	Arder To, 30 Arder Tu, 131
	ArPTZ, 298
ArLogFileConnection, 233	Arr 12, 298 ArSonyPTZ, 472
ArSerialConnection, 424	ArVCC4, 507
ArTcpConnection, 481	,
ourInitialized	PANTILTDCCW
ArSocket, 464	AramptuCommands, 99
Dod. I.I. Mar	PANTILTDCW
P2ArmJoint, 515	ArAMPTUCommands, 99
packetHandler	$\operatorname{panTiltRel}$
ArGripper, 198	ArAMPTU, 96
ArIrrfDevice, 213	ArDPPTU, 131
ArPTZ, 302	ArPTZ, 298
ArRobot, 392	ArSonyPTZ, 472
ArVCC4, 510	ArVCC4, 507
PacketType	PANTILTREQ
ArP2Arm, 277	ArVCC4Commands, 513
PAN	PANTILTUCCW
ArDPPTUCommands, 137	ArAMPTUCommands, 99
pan	PANTILTUCW
Aramptu, 96	ArAMPTUCommands, 99
ArDPPTU, 131	park
ArPTZ, 298	ArP2Arm, 274
ArSonyPTZ, 472	parseArgs
ArVCC4, 506	ArSimpleConnector, 459
· · · · · · · · · · · · · · · · · · ·	PAUSE PAUSE
panAccel	
ArDPPTU, 133	ArAMPTUCommands, 99
panRel	pause
Aramptu, 96	Aramptu, 97
ArdPPTU, 131	PLAYLIST
ArPTZ, 298	ArCommands, 121

pointRotate random ArMath, 239 ArMath, 238 POLLING read ArCommands, 119 ArDeviceConnection, 127 Pos ArLogFileConnection, 236 ArListPos, 230 ArSerialConnection, 428 POSE ArSocket, 465 ArTcpConnection, 485 ArArg, 105 POWER readPacket ArVCC4Commands, 513ArPTZ, 302 powerOff ArVCC4, 511 ArP2Arm, 281 readString ArNetServerConnection, 271 powerOn ArSocket, 466 ArP2Arm, 282 receiveBlobInfo preparePacket ArACTS_1_2, 92 ArVCC4, 508 receivePacket printHex ArRobotPacketReceiver, 409 ArBasePacket, 112 processEncoderPacketArSickPacketReceiver, 451 ArRobot, 369 recvFrom processIOPacket ArSocket, 462 ArRobot, 369 redoReading ArRangeBuffer, 310 processMotorPacket REGKEY ArRobot, 369 ArUtil, 499 processNewSonar REGKEY_CLASSES_ROOT ArRobot, 369 ArUtil, 499 processPacket REGKEY_CURRENT_CONFIG ArSick, 435 ArUtil, 499 processParamFile REGKEY_CURRENT_USER ArRobot, 370 ArUtil, 499 processReadings REGKEY_LOCAL_MACHINE ArSonarDevice, 468 ArUtil, 499 PTUPOS REGKEY_USERS ArCommands, 120 ArUtil, 499 **PULSE** regMotPowerArCommands, 119 Ardertu, 133 **PURGE** regStatPowerArAMPTUCommands, 99 ArDPPTU, 132 purge reload Aramptu, 97 ArModuleLoader, 265 RELPANCCW QUERYTYPE ArAMPTUCommands, 99 ArGripper, 198 RELPANCW radToDeg ArAMPTUCommands, 99 ArMath, 241 RELTILTD

RELTILTU ArAMPTUCommands, 99 ArP2Arm, 283 remAction RESET ArActionGroup, 61 ArDPPTUCommands, 137 ArRobot, 393 reset remCommand ArActionDesired, 51 ArNetServer, 270 ArInterpolation, 210 remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 ArSick, 442 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 130 resetPan ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 resetSensorPosition remFailedConnectCB resetSensorPosition ArRickyHandler ArSick, 442 remKeyHandler resetSensorPosition ArKeyBandler, 222 resetTilt removeArg ArResolver, 325 ArArgumentBuilder, 106 resolve remPacketHandler ArRobot, 394 ArRobot, 394	ArAMPTUCommands, 99	requestStatus
ArAMPTUCommands, 99 ArP2Arm, 283 remAction RESET ArActionGroup, 61 ArDPPTUCommands, 137 ArRobot, 393 reset remCommand ArActionDesired, 51 ArNetServer, 270 ArInterpolation, 210 remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArSick, 442 resetCalib remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSessorReading, 422 RESETSIMTOORIGIN ArCommands, 121 remKeyHandler ArCommands, 121 removeArg ArResolver, 325 ArArgumentBuilder, 106 resolve remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRunExitCB ArKeyH		-
remAction RESET ArActionGroup, 61 ArDPPTUCommands, 137 ArRobot, 393 reset remCommand ArActionDesired, 51 ArNetServer, 270 ArInterpolation, 210 remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetPan ArRobot, 394 ArDPPTU, 130 ArSick, 442 resetPan remErrorCB ArBasePacket, 117 ArSick, 442 resetRead remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 resetTilt resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394 ArSensorInterpTask <t< td=""><td></td><td></td></t<>		
ArActionGroup, 61 ArDPPTUCommands, 137 ArRobot, 393 reset remCommand ArActionDesired, 51 ArNetServer, 270 ArInterpolation, 210 remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetCalib ArRobot, 394 ArDPPTU, 130 ArBobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remErlailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 remKeyHandler ArCommands, 121 remKeyHandler ArCommands, 121 resetTilt arPPPTU, 131 ArKeyHandler, 222 resetTilt removeActions ArBopPTU, 131 ArRobot, 394 RESP ArRobot, 394 RESPONSE ArRobot, 394 RESPONSE ArRobot, 394 RESPONSE		
ArRobot, 393 reset remCommand ArActionDesired, 51 ArNetServer, 270 ArInterpolation, 210 remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetCalib ArRobot, 394 ArDPPTU, 130 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 RESETSIMTOORIGIN ArCommands, 121 remKeyHandler ArCommands, 121 remKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArRampTUCommands, 100 ArRobot, 394 RESP remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore		
remCommand ArActionDesired, 51 ArNetServer, 270 ArInterpolation, 210 remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetPan ArRobot, 394 ArDPPTU, 130 ArBick, 442 resetPan ArBobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 resetVermoveActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remRangeDevice ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArKeyHandler, 220 ArRobot, 395 restore		
ArNetServer, 270 ArInterpolation, 210 remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetCalib ArRobot, 394 ArDPPTU, 130 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 remKeyHandler ArCommands, 121 remKeyHandler, 222 resetTilt removeActions ArBPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRuneExitCB ArKeyHandler, 220 restore ArKeyHandler, 220 remSensorInterpTask ArDPPT		
remConnectCB ArRangeBuffer, 306 ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetCalib ArSick, 442 ArDPPTU, 130 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArAngenentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArKeyHandler, 220 ArRobot, 394, 395 restore remSensorInterpTask ArDPPTU, 131 <td></td> <td></td>		
ArRobot, 393 ArRecurrentTask, 323 ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetCalib ArSick, 442 resetPan remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorDosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 resetTilt ArCommands, 121 resetTilt ArBesolver, 325 removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVC4Commands, 513 ArRobot, 394, 395 restore remEastore ArKeyHandler, 220 remSensorInterpTask ArDPPTU, 131 ArRobot, 395, 396 resume <td></td> <td>- · · · · · · · · · · · · · · · · · · ·</td>		- · · · · · · · · · · · · · · · · · · ·
ArSick, 442 resetAll remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetCalib ArSick, 442 ArDPPTU, 130 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 resetVen ArDPPTU, 131 resolve resolve removeActions ArBopty, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRunexitCB ArVCC4Commands, 513 ArRobot, 394, 395 restore remSensorInterpTask ArBopPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArP2Arm, 282 RIGHT reques		
remDisconnectNormallyCB ArDPPTU, 131 ArRobot, 393 resetCalib ArSick, 442 ArDPPTU, 130 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ARSObot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRunExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remUserTask ArTaskState, 480 ArRobot, 395, 396 resume <td></td> <td></td>		
ArRobot, 393 resetCalib ArSick, 442 ArDPPTU, 130 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 resetTilt resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394 restore remSensorInterpTask ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480		
ArSick, 442 ArDPPTU, 130 remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 remKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRumExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArP2Arm, 282 RIGHT		
remDisconnectOnErrorCB resetPan ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArsgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRumExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArAmPTU, 97 ArP2Arm, 282 RIGHT requestInfo ArKeyHandler, 221		
ArRobot, 394 ArDPPTU, 131 ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRumExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArKeyHandler, 221 ArP2Arm, 282 right		
ArSick, 442 resetRead remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394 RESPONSE remRunExitCB ArKeyHandler, 220 ArRobot, 394, 395 restore remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right reques		
remErrorCB ArBasePacket, 117 ArVCC4, 507 ArSickPacket, 449 remFailedConnectCB resetSensorPosition ArRobot, 394 ArSensorReading, 422 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394 restore remRunExitCB ArKeyHandler, 220 ArRobot, 394, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 RIGHT <		
ArVCC4, 507 remFailedConnectCB ArRobot, 394 ArSick, 442 ArSick, 442 RESETSIMTOORIGIN ArKeyHandler ArKeyHandler, 222 removeActions ArActionGroup, 60 remPacketHandler ArRobot, 394 ArSensorReading, 422 RESETSIMTOORIGIN ArCommands, 121 resetTilt resolve removeArg ArActionGroup, 60 removeArg ArArgumentBuilder, 106 remPacketHandler ArRobot, 394 RESP remRangeDevice ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 100 ArRobot, 394, 395 restore remRunExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 RESUME remUserTask ArRobot, 395 RESUME requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArP2Arm, 282 requestPacket ArActionKeydrive, 75 ArACTS-1-2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
remFailedConnectCB ArRobot, 394 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArKeyHandler, 222 removeActions ArActionGroup, 60 remOveArg ArArgumentBuilder, 106 remPacketHandler ArRobot, 394 remRangeDevice ArRobot, 394 ArRobot, 395 remRunExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 requestInfo ArP2Arm, 282 requestPacket ArACTS-1-2, 92 requestQuit RESETSIMTOORIGIN ArSensorReading, 422 RESETSIMTOORIGIN ArCommands, 121 resetTilt resetTilt resetTilt RESPOVE ArACTS-1-2, 92 resetTilt ArCommands, 121 resetTilt RESPOVE ArAMPTUCommands, 100 RESPONSE ArAMPTUCommands, 513 restore ArAMPTUCommands, 513 restore ArAMPTUCommands, 513 restore ArAMPTU, 131 RESUME ArToPPTU, 131 RESUME ArTaskState, 480 resume ArAMPTU, 97 RIGHT ArKeyHandler, 221 right ROBOT_NOT_SETUP ArP2Arm, 277		•
ArRobot, 394 ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArKeyHandler, 222 resetTilt removeActions ArActionGroup, 60 removeArg ArArgumentBuilder, 106 remPacketHandler ArRobot, 394 remRangeDevice ArRobot, 395 remRunExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 requestInfo ArP2Arm, 282 requestPacket ArACTS_1.2, 92 requestQuit ArP2Arm, 277 RESETSIMTOORIGIN RESETSIMTOORIGIN RESETSIMTOORIGIN RESETSIMTOORIGIN ArCommands, 121 resetTilt resetTilt resetTilt resetTilt resetTilt RESPONSE RESP RESPONSE ArAMPTUCommands, 100 RESPONSE ArACTSALSALSALSALSALSALSALSALSALSALSALSALSALS	•	,
ArSick, 442 RESETSIMTOORIGIN remKeyHandler ArCommands, 121 ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRunExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1.2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
remKeyHandler ArKeyHandler, 222 resetTilt removeActions ArActionGroup, 60 removeArg ArArgumentBuilder, 106 ArRobot, 394 remRangeDevice ArRobot, 395 remRunExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 requestInfo ArRobot, 395 requestInfo ArP2Arm, 282 requestPacket ArACTS_1_2, 92 requestQuit ArP2Arm, 277 resolve resetTilt resetTilt resetTilt resetTilt resetTilt resetTilt resetTilt resetTilt resetTilt reselve RARDPTU, 131 ArCommands, 100 RESP RESPONSE ArAMPTUCommands, 100 RESPONSE ArAMPTUCommands, 100 RESPONSE ArAMPTUCommands, 100 RESPONSE ArAMPTUCommands, 100 ArAMPTUCommands, 100 RESPONSE ArAMPTUCommands, 100 ArAMPTUCommands, 100 RESPONSE ArAMPTUCommands, 100 ArAMPTUCommands, 100 ArAMPTUCommands, 100 ArAMPTUCommands, 100 ArAMPTUCommands, 100 ArACTSALOR ARA		
ArKeyHandler, 222 resetTilt removeActions ArDPPTU, 131 ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRunExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 right ArActionKeydrive, 75 ArACTS_1.2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
removeActions ArActionGroup, 60 removeArg ArArgumentBuilder, 106 remPacketHandler ArRobot, 394 remRangeDevice ArRobot, 395 remRumExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 reguestInfo ArP2Arm, 282 requestPacket ArACTS_1.2, 92 requestQuit ArP2Arm, 277 ArP2Arm, 277 ArP2Arm, 277 ArP2Arm, 277 ArP2Arm, 282 RIGHT resolve resolve RAROBOT, 325 RESPONSE RESPONSE ArAMPTUCommands, 100 ArACHOCHOMMANDS ArA		
ArActionGroup, 60 resolve removeArg ArResolver, 325 ArArgumentBuilder, 106 RESP remPacketHandler ArAMPTUCommands, 100 ArRobot, 394 RESPONSE remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRunExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
removeArg ArArgumentBuilder, 106 RESP remPacketHandler ArRobot, 394 RESPONSE remRangeDevice ArRobot, 394, 395 restore remRunExitCB ArRobot, 395 restoreSet remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 resume requestInfo ArRobot, 395 resume requestInfo ArP2Arm, 282 requestInit ArP2Arm, 282 requestPacket ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
ArArgumentBuilder, 106 remPacketHandler ArRobot, 394 remRangeDevice ArRobot, 394, 395 remRunExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 requestInfo ArP2Arm, 282 requestPacket ArP2Arm, 282 requestPacket ArACTS_1_2, 92 remPacket ArAMPTU ArP2Arm, 277 ArP2Arm, 277 ArP2Arm, 277 RESP ArAMPTU Commands, 100 RESP ArAMPTUCommands, 100 RESP ArAMPTUCommands, 100 RESP ArAMPTUCommands, 100 RESP RESP ArAMPTUCommands, 100 ArAMPTUCommands, 100 ArAMPTUCommands, 100 RESP ArAMPTUCommands, 100 ArACHAMPTU, 21 ArAMPTU, 97 RIGHT ArACTIONE ArAC		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	~	
ArRobot, 394 remRangeDevice ArRobot, 394, 395 remRunExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395 reguestInfo ArP2Arm, 282 requestPacket ArP2Arm, 282 requestPacket ArACTS_1_2, 92 requestQuit ArPCC4Commands, 513 RESUOR ArVCC4Commands, 513 RESUOR ArKeyHandler, 220 restoreSet ArKeyHandler, 220 resume resume ArAMPTU, 97 RIGHT RIGHT ArKeyHandler, 221 right ROBOT_NOT_SETUP ROBOT_NOT_SETUP ArP2Arm, 277		
remRangeDevice ArVCC4Commands, 513 ArRobot, 394, 395 restore remRunExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
ArRobot, 394, 395 restore remRunExitCB ArKeyHandler, 220 ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
remRunExitCB ArRobot, 395 remSensorInterpTask ArRobot, 395 remUserTask ArRobot, 395 remUserTask ArRobot, 395, 396 requestInfo ArRobot, 395, 396 requestInfo ArP2Arm, 282 requestInit ArP2Arm, 282 requestPacket ArACTS_1_2, 92 requestQuit ArP2Arm, 277 ArRobot, 395, 396 RESUME ArTaskState, 480 resume ArAMPTU, 97 ArAMPTU, 97 RIGHT ArKeyHandler, 221 right ROBOT_NOT_SETUP ROBOT_NOT_SETUP ArP2Arm, 277		ArVCC4Commands, 513
ArRobot, 395 restoreSet remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
remSensorInterpTask ArDPPTU, 131 ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
ArRobot, 395 RESUME remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
remUserTask ArTaskState, 480 ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277	-	
ArRobot, 395, 396 resume requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
requestInfo ArAMPTU, 97 ArP2Arm, 282 RIGHT requestInit ArKeyHandler, 221 ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		ArTaskState, 480
ArP2Arm, 282 requestInit ArP2Arm, 282 right requestPacket ArACTS_1_2, 92 requestQuit RIGHT ArKeyHandler, 221 right ArActionKeydrive, 75 ROBOT_NOT_SETUP ArP2Arm, 277	· · · · · · · · · · · · · · · · · · ·	
requestInit ArKeyHandler, 221 ArP2Arm, 282 requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 requestQuit ArP2Arm, 277	1	•
ArP2Arm, 282 right requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277		
requestPacket ArActionKeydrive, 75 ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277	requestInit	ArKeyHandler, 221
ArACTS_1_2, 92 ROBOT_NOT_SETUP requestQuit ArP2Arm, 277	ArP2Arm, 282	right
requestQuit ArP2Arm, 277	•	• • • • • • • • • • • • • • • • • • • •
		ROBOT_NOT_SETUP
ArACTS_1_2, 92 robotConnectCallback	requestQuit	
,	ArACTS_1_2, 92	${\bf robotConnectCallback}$

Angiala 425	ArTimo 401
ArSick, 435 robotLocker	ArTime, 491 self
	ArThread, 490
ArRobot, 396 robotPacketHandler	sendPacket
ArPTZ, 302	ArPTZ, 303
robotTask	sendTo
ArSickLogger, 444	ArSocket, 462
robotUnlocker	sendToAllClients
ArRobot, 396	ArNetServer, 270
ROTATE	sendToAllClientsPlain
ArCommands, 120	ArNetServer, 268
roundInt	sensorInterpCallback
ArMath, 241	ArSick, 435
run	sensorInterpHandler
ArASyncTask, 110	ArPTZ, 300
ArRangeDeviceThreaded, 320	SETA
ArRobot, 396	ArCommands, 119
ArSyncTask, 479	setArea
runAsync	ArACTSBlob, 94
ArASyncTask, 110	setAutoParkTimer
ArRangeDeviceThreaded, 320	ArP2Arm, 283
ArRobot, 396	setAuxPort
runInThisThread	ArPTZ, 303
ArASyncTask, 111	setBaud
runOnce	ArSerialConnection, 429
ArNetServer, 268	$\operatorname{setBool}$
ArSick, 435	ArArg, 104
runOnRobot	ArPref, 294
ArSick, 443	$\operatorname{setBoolSet}$
runThread	ArPref, 294
ArASyncTask, 111	setBottom
ArFunctorASyncTask, 164	ArACTSBlob, 95
ArRangeDeviceThreaded, 320	$\operatorname{setBroadcast}$
ArRecurrentTask, 324	ArSocket, 462
ArSick, 435	setBuf
ArSignalHandler, 457	ArBasePacket, 114
RVEL	$\operatorname{setCloseDist}$
ArCommands, 120	ArActionGoto, 58
_	${\bf set Connection Cycle Multiplier}$
saveSet	ArRobot, 397
ArDPPTU, 131	setConnectionTimeoutTime
SAY	ArRobot, 397
ArCommands, 120	ArSick, 443
secSince	${\bf set Cumulative Buffer Size}$
ArTime, 491	ArRangeDevice, 318
secTo	setCumulativeMaxRange

ArIrrfDevice, 212	ArP2Arm, 283
ArSonarDevice, 468	setHardwareControl
setCurrentBufferSize	ArSerialConnection, 429
ArRangeDevice, 318	
setCycleChained	setHeaderLength ArBasePacket, 114
·	
ArRobot, 367	setHeading
setCycleTime	ArActionDesired, 55
ArRobot, 397	ArRobot, 400
setDeadReconPose	setHeadingDoneDiff
ArRobot, 398	ArRobot, 357
setDegDiff	setID
ArSickLogger, 444	ArRobotPacket, 406
setDeltaHeading	setIncrements
ArActionDesired, 54	ArActionKeydrive, 74
ArRobot, 398	setInt
setDeviceConnection	ArArg, 104
ArPTZ, 303	ArPref, 295
ArRobot, 398	setIntSet
ArRobotPacketReceiver, 408	ArPref, 296
ArRobotPacketSender, 411	setKeyHandler
ArSick, 432	Aria, 206
ArSickPacketReceiver, 450	$\operatorname{setLeft}$
${\bf set Direct Motion Precedence Time}$	ArACTSBlob, 95
ArRobot, 398	$\operatorname{setLength}$
setDirectory	ArBasePacket, 114
Aria, 208	$\operatorname{setLinger}$
$\operatorname{setDistDiff}$	ArSocket, 462
ArSickLogger, 444	$\operatorname{setMaxNegVel}$
setDoClose	ArActionDesired, 55
ArSocket, 463	setMaxRange
setDouble	ArRangeDevice, 314
ArArg, 104	$\operatorname{setMaxRotVel}$
ArPref, 295	ArActionDesired, 55
setDoubleSet	ArRobot, 400
ArPref, 295	$\operatorname{setMaxTransVel}$
setEcho	ArRobot, 400
ArNetServerConnection, 271	setMaxVel
setEncoderCorrectionCallback	ArActionDesired, 55
ArRobot, 399	setMinRange
setEncoderTransform	ArSick, 432
ArRobot, 399	setMoveDoneDist
setFilterNearDist	ArRobot, 357
ArSick, 443	setMSec
setGoal	
ArActionGoto, 58	ArTime, 492 setName
setGripperParkTimer	
	ArRobot, 362

setNextArgument	SETRA
ArAction, 41	ArCommands, 120
setNonBlock	SETRANGE
ArSocket, 463	ArVCC4Commands, 513
setNumberOfReadings	setReadLength
ArInterpolation, 210	ArBasePacket, 114
SETO	setResolver
ArCommands, 119	ArRobot, 366 setReuseAddress
setP1	
ArFunctor1C, 146	ArSocket, 463
ArFunctor2C, 153	setRight
ArFunctor3C, 162	ArACTSBlob, 95
ArGlobalFunctor1, 171	setRobot
ArGlobalFunctor2, 175	ArAction, 41
ArGlobalFunctor3, 180	ArActionKeydrive, 74
ArGlobalRetFunctor1, 186	ArACTS_1_2, 89
ArGlobalRetFunctor2, 189	ArIrrfDevice, 212
ArGlobalRetFunctor3, 194	ArModule, 260
ArRetFunctor1C, 332	ArP2Arm, 273
ArRetFunctor2C, 340	ArRangeDevice, 312
ArRetFunctor3C, 350	ArSick, 435
setP2	ArSonarDevice, 468
ArFunctor2C, 153	$\operatorname{setRotVel}$
ArFunctor3C, 162	ArRobot, 401
ArGlobalFunctor2, 175	$\operatorname{setRunning}$
ArGlobalFunctor3, 180	ArThread, 488
ArGlobalRetFunctor2, 189	SETRV
ArGlobalRetFunctor3, 194	ArCommands, 120
ArRetFunctor2C, 340	setSec
ArRetFunctor3C, 350	ArTime, 492
setP3	$\operatorname{setSendingAddress}$
ArFunctor3C, 162	ArSickPacket, 449
ArGlobalFunctor3, 181	setSensorPosition
ArGlobalRetFunctor3, 195	ArSick, 431, 432
ArRetFunctor3C, 350	SETSIMORIGINTH
$\operatorname{setPacketCB}$	ArCommands, 121
ArP2Arm, 275	SETSIMORIGINX
setPort	ArCommands, 121
ArSerialConnection, 429	SETSIMORIGINY
setPose	ArCommands, 121
ArArg, 104	$\operatorname{setSize}$
ArPose, 288	ArRangeBuffer, 311
setPoseInterpNumReadings	setSocket
ArRobot, 367	ArTcpConnection, 485
setPoseTaken	setSpeed
ArRangeBuffer, 305	ArActionGoto, 58
-11100110020 011011, 0000	111110110110000, 00

setSpeeds	ArAMPTUPacket, 102
ArActionJoydrive, 70	setupLaser
ArActionKeydrive, 74	ArSimpleConnector, 460
ArJoyHandler, 214	setUpPacketHandlers
setState	ArRobot, 369
ArSyncTask, 475	setupRobot
setStateReflectionRefreshTime	ArSimpleConnector, 460
ArRobot, 401	setUpSyncList
setStats	ArRobot, 369
ArJoyHandler, 215	setUseOSCal
setStatus	ArActionJoydrive, 72
ArTcpConnection, 482	ArJoyHandler, 218
setStopIfNoButtonPressed	SETV
ArActionJoydrive, 70	ArCommands, 119
setStoppedCB	setVel
ArP2Arm, 275	ArActionDesired, 56
	ArActionInput, 68
setString ArArg, 104	ArRobot, 401
	•
ArPref, 296	setVel2
setTh	ArRobot, 402
ArPose, 285	setX
setThis	ArPose, 285
ArFunctor1C, 146, 147	setXCG
ArFunctor2C, 154	ArACTSBlob, 94
ArFunctor3C, 162, 163	setY
ArFunctorC, 166	ArPose, 285
ArRetFunctor1C, 333	setYCG
ArRetFunctor2C, 340	ArACTSBlob, 94
ArRetFunctor3C, 350, 351	shutdown
ArRetFunctorC, 353	Aria, 209
setThRad	ArSocket, 466
ArPose, 285	SIGHANDLE_NONE
setThrottleParams	Aria, 206
ArActionJoydrive, 70	SIGHANDLE_SINGLE
setTimeReceived	Aria, 206
ArRobotPacket, 406	SIGHANDLE_THREAD
ArSickPacket, 448	Aria, 206
setToNow	SigHandleMethod
ArTime, 491	Aria, 206
setTop	signal
ArACTSBlob, 95	ArCondition, 122
setTransform	signalHandlerCB
ArTransform, 495	Aria, 206
setType	simPacketHandler
ArGripper, 202	ArSick, 435
setUnitNumber	sin

ArMath, 241 ArSick, 437 STATE_INSTALL_MODE sizeFile ArUtil, 502 ArSick, 437 slaveExec STATE_NONE Arden ArSick, 437 sleep STATE_SET_MODE ArUtil, 502 ArSick, 437 STATE_START_READINGS SLEWREQ ArVCC4Commands, 513 ArSick, 437 sockAddrIn STATE_WAIT_FOR_-ArSocket, 462 CONFIGURE_ACK sockAddrLenArSick, 437 ArSocket, 464 STATE_WAIT_FOR_INSTALL_-SONAR MODE_ACK ArCommands, 120 ArSick, 437 SOUND STATE_WAIT_FOR_POWER_ON ArCommands, 121 ArSick, 437 SOUNDTOG STATE_WAIT_FOR_SET_MODE_-ArCommands, 121 ACK SPACE ArSick, 437 STATE_WAIT_FOR_START_ACK ArKeyHandler, 221 space ArSick, 437 ArActionKeydrive, 75 stateReflector SPEED ArRobot, 402 ArDPPTUCommands, 137 STATUS splitString ArAMPTUCommands, 99 ArUtil, 503 Status squaredDistanceBetween ArDeviceConnection, 125 ArMath, 242 ArModuleLoader, 264 squaredFindDistanceTo ArMutex, 267 ArPose, 288 ArThread, 490 startAverage STATUS_ALREADY_-ArActionDesired, 56 DETATCHED ArThread, 490 startCal ArJoyHandler, 219 STATUS_ALREADY_LOADED State ArModuleLoader, 264 ArP2Arm, 277 STATUS_ALREADY_LOCKED ArSick, 437 ArMutex, 267 ArTaskState, 480 STATUS_CLOSED_ERROR STATE_CHANGE_BAUD ArDeviceConnection, 125 ArSick, 437 STATUS_CLOSED_NORMALLY STATE_CONFIGURE ArDeviceConnection, 125 ArSick, 437 STATUS_EXIT_FAILED STATE_CONNECTED ArModuleLoader, 264 ArSick, 437 STATUS_FAILED STATE_INIT ArCondition, 123

ArMutex, 267 StatusType ArP2Arm, 278 ArThread, 490 STATUS_FAILED_DESTROY StdErr ArCondition, 123 ArLog, 232 STATUS_FAILED_INIT StdOut ArCondition, 123 ArLog, 232 ArMutex, 267 STEP STATUS_FAILED_OPEN ArCommands, 120 ArModuleLoader, 264 STOP STATUS_INIT_FAILED ArCommands, 120 ArModuleLoader, 264 ArVCC4Commands, 513 STATUS_INVALID stop ArModuleLoader, 264 ArP2Arm, 283 ArThread, 490 ArRobot, 402 STATUS_JOIN_SELF stopAll ArThread, 490 ArThread, 489 STATUS_MUTEX_FAILED stopRunning ArCondition, 123 ArASyncTask, 110 STATUS_MUTEX_FAILED_INIT ArRangeDeviceThreaded, 320 ArCondition, 123 ArRobot, 403 STATUS_NEVER_OPENED ArThread, 487 ArDeviceConnection, 125 strcmp STATUS_NO_SUCH_THREAD ArUtil, 503, 504 ArThread, 490 STRING STATUS_NORESOURCE ArArg, 105 ArThread, 490 String STATUS_NOT_FOUND ArPref, 292 ArModuleLoader, 264 stripDir STATUS_OPEN ArUtil, 505 ArDeviceConnection, 125 stripFile ${\tt STATUS_OPEN_FAILED}$ ArUtil, 505 ArDeviceConnection, 125 strNToBuf STATUS_SUCCESS ArBasePacket, 117 ArModuleLoader, 264 strToBuf STATUS_WAIT_INTR ArBasePacket, 117 ArCondition, 123 strToBufPadded STATUS_WAIT_TIMEDOUT ArBasePacket, 117 ArCondition, 123 subAngle ArMath, 242 StatusContinuous ArP2Arm, 278 SUCCESS StatusOff ArP2Arm, 277 ArP2Arm, 278 ArTaskState, 480 StatusPacket SUSPEND ArP2Arm, 277 ArTaskState, 480 StatusSingle switchState ArP2Arm, 278 ArSick, 436

TAB	ArThread, 488
ArKeyHandler, 221	tryLockDevice
takeKeys	ArRangeDevice, 319
ArActionKeydrive, 75	ArRangeDeviceThreaded, 321
takeReading	TTY2
ArSickLogger, 444	ArCommands, 120
task	Type
ArRecurrentTask, 324	ArArg, 105
TCM2	ArGripper, 198
ArCommands, 120	typedef
Terse	ArCondition, 123
ArLog, 231	Arcondition, 125
TILT	UByte
ArDPPTUCommands, 137	ArTypes, 496
tilt	UByte2
ArAMPTU, 96	ArTypes, 496
	uByte2ToBuf
ArDPPTU, 131	ArBasePacket, 113
ArPTZ, 298	UByte4
ArSonyPTZ, 472	
ArVCC4, 506	ArTypes, 496 uByte4ToBuf
tiltAccel	ArBasePacket, 113
Ardpptu, 133	
tiltRel	uByteToBuf
Aramptu, 96	ArBasePacket, 113
Ardpptu, 131	ArSonyPacket, 470
ArPTZ, 298	unblock
ArSonyPTZ, 472	ArSignalHandler, 457
ArVCC4, 507	unblockAll
TILTSLEW	ArSignalHandler, 458
ArAMPTUCommands, 100	unhandle
ArVCC4Commands, 513	ArSignalHandler, 458
tiltSlew	uninit
ArAMPTU, 97	Aria, 209
ArDPPTU, 134	ArP2Arm, 284
ArVCC4, 508	unlock
tiltSlewRel	ArMutex, 266
ArDPPTU, 134	ArRobot, 368
timedWait	ArThread, 488
ArCondition, 122	unlockDevice
transfer	ArRangeDevice, 319
ArSocket, 466	ArRangeDeviceThreaded, 321
trying To Connect	UP
ArSick, 432	ArKeyHandler, 221
tryLock	up
ArMutex, 267	ArActionKeydrive, 75
ArRobot, 368	update

ArSectors, 418	WAIT_TIMEDOUT
UPPER	ArRobot, 370
ArDPPTUCommands, 137	waitForConnect
upperPanSlew	ArRobot, 403
ArDPPTU, 133	wait For Connect Or Conn Fail
upperTiltSlew	ArRobot, 403
ArDPPTU, 133	waitForRunExit
USER_START	ArRobot, 404
ArTaskState, 480	WaitState
USERIO	ArRobot, 370
ArGripper, 198	wake All Conn Or Fail Waiting Threads
userTask	ArRobot, 404
ArMode, 245	wakeAllConnWaitingThreads
ArModeCamera, 248	ArRobot, 404
ArModeGripper, 250	wakeAllRunExitWaitingThreads
ArModeSonar, 252	ArRobot, 405
	wakeAllWaitingThreads
ValType	ArRobot, 405
ArPref, 292	wasError
VEL	ArVCC4, 509
ArCommands, 120	write
VEL2	ArDeviceConnection, 128
ArCommands, 120	ArLogFileConnection, 237
velMove	ArSerialConnection, 429
ArDPPTU, 133	ArSocket, 467
VELOCITY	ArTcpConnection, 485
ArDPPTUCommands, 137	writePacket
Verbose	ArDeviceConnection, 128
ArLog, 232	writeString
verifyCheckSum	ArSocket, 467
ArRobotPacket, 406	writeStringPlain
verifyCRC	ArSocket, 463
ArSickPacket, 447	THE COLOU, 100
,	yieldProcessor
wait	ArThread, 489
ArCondition, 122	,
WAIT_CONNECTED	ZOOM
ArRobot, 370	ArAMPTUCommands, 99
WAIT_FAIL	ArVCC4Commands, 513
ArRobot, 370	zoom
WAIT_FAILED_CONN	ArPTZ, 299
ArRobot, 370	ArSonyPTZ, 473
WAIT_INTR	ArVCC4, 507
ArRobot, 370	zoomRel
WAIT_RUN_EXIT	ArPTZ, 299
ArRobot, 370	ArSonyPTZ, 473
	1110011,110

 $\begin{array}{c} {\rm ZOOMSTOP} \\ {\rm ArVCC4Commands}, \, 513 \end{array}$